N° d'ordre : 4063 ANNÉE 2009

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Européenne de Bretagne

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Mention : INFORMATIQUE

Ecole doctorale MATISSE

présentée par

Boris Daix

préparée à l'unité de recherche 6074 IRISA
Institut de Recherche en Informatique et Systèmes Aléatoires
(IFSIC)

Abstraction des systèmes informatiques à haute performance pour l'automatisation du déploiement d'applications dynamiques

Thèse soutenue à Rennes
le 11 décembre 2009

devant le jury composé de :

Thierry Priol
Directeur de recherche, INRIA Rennes / président

Daniel Hagimont
Professeur, ENSEEIHT / rapporteur

Philippe Merle
Chargé de recherche, INRIA Lille / rapporteur

Marco Danelutto
Professeur associé, Université de Pise / examinateur

Christine Morin
Directrice de recherche, INRIA Rennes / directeur de thèse

Christian Pérez
Chargé de recherche, INRIA Grenoble / co-directeur de thèse

Samuel Kortas
Ingénieur-chercheur, EDF R&D / co-directeur de thèse

Mes remerciements vont à MM. Hagimont et Merle, rapporteurs, à M. Danelutto, examinateur, ainsi qu'à M. Priol, président du jury. Merci à vous pour vos critiques, vos questions, et votre enthousiasme concernant cette thèse. Je remercie également les membres de l'équipe qui a dirigé mes travaux, M^me^ Morin ainsi que MM. Kortas et Pérez. Attentifs, vous avez animé le partenariat EDF R&D — INRIA Rennes dont cette thèse faisait l'objet et avez assuré les conditions qui m'ont conduit jusqu'au titre. Merci Christine pour le temps que tu as pu m'accorder, Samuel pour tes explications et ton optimisme, et Christian pour la force que tu m'as transmise. Merci à vous aussi, membres de l'atelier à IRISA et membres du projet SALOME à EDF R&D, pour votre patience et pour vos précieux conseils.

Je remercie également tous ceux qui m'ont transmis la passion de l'informatique et qui m'ont permis d'en faire mon métier. Merci à vous, MM. Ollier, Chaunier, et Girard d'avoir initié le collégien puis le lycéen que j'étais dans les années 1990. Merci à vous, MM. Pollet et Emptoz du département Premier Cycle, ainsi qu'à tous les membres de l'équipe de M. Pinon du département Informatique, à l'INSA Lyon. Vous avez fait de moi l'ingénieur INSA que je suis avant tout. Merci également à vous, membres des équipes de M. Villard du département Mathématiques et Informatique, à l'ENS Lyon, et de M. Priol du projet PARIS, à l'INRIA Rennes.

Pour m'avoir accueilli et formé dans vos prestigieuses entreprises, merci à vous, membres de l'équipe de M. Arias des *Global Deployment Services* chez Hewlett-Packard, membres de l'équipe de M. Iffenecker du SEPTEN chez EDF, et membres des équipes de MM. Berthou, Gayrard, et Bateman de SINETICS chez EDF R&D. J'ai apprécié collaborer avec vous et vous dois mes qualités relationnelles, indispensables et pas seulement dans la vie active.

Je salue l'œuvre remarquable des nombreuses personnes, agissant en faveur de l'inclusion des personnes en situation de handicap, que j'ai eu l'honneur de rencontrer dans mon parcours scolaire. Merci à vous, membres de l'équipe de l'école adaptée EREADV, membres de l'équipe de l'*International Computer Camp*, membres des associations AGEFIPH, FIDEV, ALLP, ADEM, et AIR de Bretagne, membres de l'équipe de M. Charlin à la Mission Handicap de l'université Claude Bernard Lyon I, M. Farcy du CNRS ainsi que M^me^ et M. Guillet-Carle du district 103CS du Lions Club, coordinateurs handicap M. Balsollier et M^me^ Rumpler de l'INSA Lyon, assistant M. Schwebel, coopérateurs de La Construction Arbresloise, membres du service social de l'Union des SCOPs, et membres des missions handicap des entreprises Hewlett-Packard et EDF. J'ai une pensée particulière pour feu M. Claude Décoret, directeur de recherche au CNRS et président de la Mission Handicap de Lyon I, qui non seulement a constitué un formidable exemple pour moi mais qui a aussi veillé sur mon parcours : Claude, tu nous manques beaucoup.

Je salue également la démarche, non moins remarquable, des membres des différentes communautés du Logiciel Libre et de la Connaissance Libre. Grâce à vous, contributeurs des projets GNU, Linux, Debian, LaTeX, BRLTTY, Gutenberg.org, et Wikipedia notamment ainsi que vous, membres fondateurs du Groupe des Pingouins Libres à l'INSA Lyon, j'ai pu surmonter la plupart des situations de handicap dans lesquelles m'abandonnaient l'informatique grand public, d'une part, et les solutions privatrices d'assistance technologique au handicap, d'autre part. Parce que vos produits sont ouverts et que vous êtes attentifs à la diversité de leurs utilisateurs, j'ai pu les maîtriser et me développer avec eux jusqu'à obtenir le titre.

J'embrasse amicalement tous ceux qui m'ont supporté, au travail ou dans la détente, pour le meilleur ou pour le pire. Merci à vous, MM. Lottiaux, Gallard, Utard, Fertré, Rilling, Jeanvoine, Parpaillon, Raoult, Collin, Leprince, et Prisker ainsi que M^lles^ Bouziane, Drémeau, et Vergerio. Merci aux membres du projet SYMBIOSE qui avez accueilli, dans votre couloir du bâtiment 12D, à

l'IRISA, le “réfugié climatique” en provenance du bâtiment 12E que j'étais. Merci également à tous les membres de la *batucada* de l'IRISA : jouer de la caisse claire dans une fanfare, en dansant les yeux fermés et avec des bouchons dans les oreilles, fut une expérience inoubliable !

J'embrasse tendrement cette fois M^lle^ Sarah Quaresimin, ma compagne, et M^lle^ Célestine Daix, notre fille. Sarah, nous avons traversé ensemble cette formidable aventure et je te dois d'abord d'y être entré puis d'en être ressorti. Parce que les mots me manquent, je reprendrai l'expression de la physicienne et épistémologue M^me^ Miora Mugur-Schächter, remerciant son mari “qui, cette fois aussi —- comme toujours —- a constamment protégé avec patience et force les conditions fragiles de mes éloignements dans l'abstrait”. Célestine, je te remercie de m'avoir encouragé à terminer ce travail, de tes petits coups de pieds, blottie dans le ventre de ta mère. Je te remercie également d'avoir attendu trois jours, après ma soutenance, avant de venir au monde pour notre plus grande joie, le 15 décembre 2009. J'embrasse également ma famille et mes amis d'avoir souffert patiemment de mon éloignement et de ma récurrente indisponibilité : vous m'avez manqué !

Je dédie ce travail à feu M. Anthony Longefay, mon cousin germain, disparu à la fin de l'été 1996, ainsi qu'à feu M. Matthieu Fayadat, mon meilleur ami de collège, disparu en début d'année 2000. Mes frères, votre souvenir étincelant m'accompagne dans chacun des événements qui jalonnent ma vie.

Table des matières

I Introduction **13**

1 Introduction **15**
1.1 L'informatique : formidable levier du développement humain 15
1.2 Des systèmes informatiques à haute performance difficiles à utiliser 15
1.3 Vers l'automatisation de l'utilisation des systèmes informatiques à haute performance 16
1.4 Abstraction des systèmes informatiques à haute performance pour l'automatisation du déploiement d'applications dynamiques 17
1.5 Organisation du document 18

II Contexte et problématique **19**

2 Informatique à haute performance **21**
2.1 Introduction 21
2.2 Principes 21
2.2.1 Structure générale de l'informatique 22
2.2.2 Facteurs d'amélioration des performances 24
2.3 Ressources, applications, et systèmes d'exploitation 25
2.3.1 Ressources 25
Architectures de ressources 27
Mémoires, processeurs, et connecteurs 29
Discussion 31
2.3.2 Applications 31
Modèles d'applications 31
Données, traitements, et flux 34
Discussion 36
2.3.3 Systèmes d'exploitation 37
Principes 37
Exemples de systèmes d'exploitation 38
Mémorisation, exécution, et adressage 39
Discussion 40
2.4 Discussion 41
2.5 Conclusion 41

3 Déploiement dans le contexte de l'informatique à haute performance **43**
3.1 Introduction 43
3.2 Cycle de vie des applications sur les ressources 43
3.2.1 Développement 43

3.2.2 Déploiement . . . 45
3.2.3 Contexte de l'informatique à haute performance . . . 47
Respect du principe de compatibilité . . . 47
Respect du principe de faisabilité . . . 48
3.3 Exemples de déploiement . . . 48
3.3.1 Application paramétrique sur ordinateur . . . 48
Cas de déploiement . . . 48
Processus de déploiement . . . 49
3.3.2 Application parallèle sur grappe d'ordinateurs . . . 50
Cas de déploiement . . . 50
Processus de déploiement . . . 51
3.3.3 Application en flot de travail sur grille d'ordinateurs . . . 51
Cas de déploiement . . . 52
Processus de déploiement . . . 52
3.3.4 Discussion . . . 53
3.4 Allocation, installation, et exécution . . . 54
3.4.1 Allocation . . . 54
Langages de description . . . 54
Systèmes d'information . . . 56
Ordonnanceurs . . . 57
3.4.2 Installation . . . 57
Images disques . . . 58
Archives . . . 58
Paquets . . . 58
Composants . . . 59
3.4.3 Exécution . . . 59
3.5 Déploiement . . . 59
3.5.1 Administration de ressources . . . 60
3.5.2 Gestion d'applications à base de composants . . . 61
CCM . . . 61
Fractal . . . 62
3.5.3 Outils de déploiement . . . 63
GoDIET . . . 63
Adage . . . 64
CORDAGE . . . 64
3.6 Discussion . . . 65
3.7 Conclusion . . . 68

III Contribution **71**

4 Architecture ODD/SAMURAAIE de déploiement dynamique d'application **73**
4.1 Introduction . . . 73
4.2 Principes . . . 73
4.2.1 Modélisation . . . 73
4.2.2 Permanence . . . 74
4.3 Modèle SAMURAAIE d'abstraction des systèmes informatiques . . . 74
4.3.1 Contenant, contenu, et association . . . 75
Éléments . . . 75
Relations . . . 77

4.3.2 Instances, actions, et événements . . . 77
Ressources, Application, et Carte . . . 77
Services, Programme, et Déploiement . . . 79
Capteurs, Messages, et Journal . . . 80
4.3.3 Exemples . . . 81
Exécution d'un processus . . . 81
Arrêt d'un processus en cours d'exécution . . . 85
Ajout d'un processeur . . . 85
4.3.4 Discussion . . . 86
4.4 Modèle ODD d'automatisation du déploiement d'applications . . . 88
4.4.1 Acteurs . . . 89
Gestionnaires . . . 89
Abstracteurs . . . 92
Traducteurs . . . 93
Connecteurs . . . 93
4.4.2 Interactions . . . 94
Classes d'acteurs . . . 95
Acteurs . . . 95
Modification d'abstraction . . . 95
Démarrage et arrêt du modèle . . . 98
4.4.3 Discussion . . . 98
4.5 Conclusion . . . 99

5 Prototype ANGE **101**
5.1 Introduction . . . 101
5.2 Généralités . . . 101
5.3 Implantation de SAMURAAIE . . . 102
5.3.1 Abstractions, éléments, et relations . . . 102
5.3.2 Caractéristiques des éléments et des relations . . . 103
5.3.3 Sauvegarde des caractéristiques dans une base relationnelle . . . 103
5.4 Implantation d'ODD . . . 104
5.4.1 Gestionnaires . . . 105
5.4.2 Abstracteurs . . . 105
5.4.3 Traducteurs . . . 105
5.4.4 Connecteurs . . . 106
5.5 Ajout du support d'une technologie . . . 106
5.6 Discussion . . . 107
5.7 Conclusion . . . 108

IV Validation **109**

6 Automatisation du déploiement d'applications parallèles, paramétriques, et en flot de travail **111**
6.1 Introduction . . . 111
6.2 Principes . . . 111
6.3 Déploiement sur ordinateurs d'applications parallèles . . . 112
6.3.1 Un ordinateur et une application parallèle . . . 112
6.3.2 Traitement avec ODD/SAMURAAIE . . . 113

Abstraction avec SAMURAAIE . . . 113
Automatisation avec ODD . . . 116
6.4 Déploiement sur grappes d'ordinateurs d'applications paramétriques . . . 120
6.4.1 Une grappe d'ordinateurs et une application paramétrique . . . 120
6.4.2 Traitement avec ODD/SAMURAAIE . . . 121
Abstraction avec SAMURAAIE . . . 121
Automatisation avec ODD . . . 123
6.5 Déploiement sur grilles d'ordinateurs d'applications en flot de travail . . . 125
6.5.1 Une grille d'ordinateurs et une application en flot de travail . . . 125
6.5.2 Traitement avec ODD/SAMURAAIE . . . 126
Abstraction avec SAMURAAIE . . . 126
Automatisation avec ODD . . . 128
6.6 Discussion . . . 129
6.7 Conclusion . . . 130

7 Automatisation du déploiement de simulations numériques SALOME 131
7.1 Introduction . . . 131
7.2 SALOME et son approche du déploiement . . . 131
7.2.1 Introduction . . . 131
7.2.2 Architecture générale . . . 132
7.2.3 Modèle de composants . . . 132
Introduction . . . 132
Objets PaCO++ . . . 134
Objets SALOME . . . 134
Composants DSC . . . 134
Schémas de calcul . . . 135
Discussion . . . 136
Conclusion . . . 136
7.2.4 Déploiement . . . 136
Introduction . . . 136
Installation . . . 136
Exécution . . . 139
Discussion . . . 143
Conclusion . . . 144
7.2.5 Conclusion . . . 144
7.3 Délégation par SALOME du déploiement . . . 145
7.3.1 Introduction . . . 145
7.3.2 Intégration de SALOME et d'ODD/SAMURAAIE . . . 145
Captation des données de déploiement par des connecteurs ODD pour SALOME . . . 147
Traduction des données de déploiement par des traducteurs ODD pour SALOME . . . 147
Fonctionnement . . . 149
7.3.3 Implantation de l'intégration de SALOME et d'ODD/SAMURAAIE . . . 150
Extension d'ANGE pour la technologie SALOME . . . 150
Modification de SALOME pour l'utilisation d'ANGE . . . 151
7.3.4 Réalisation d'une simulation numérique SALOME déployée par ANGE . . . 151
7.3.5 Discussion . . . 152
7.3.6 Conclusion . . . 152

7.4 Évolutions de SALOME pour le déploiement dynamique . . . 153
7.4.1 Introduction . . . 153
7.4.2 Allocation optimisée . . . 153
7.4.3 Installation à la volée . . . 154
7.4.4 Exécution anticipée . . . 154
7.4.5 Conclusion . . . 154
7.5 Conclusion . . . 155

V Conclusion **157**

8 Conclusion **159**
8.1 Abstraire les systèmes informatiques afin d'en automatiser l'utilisation . . . 159
8.2 Perspectives . . . 160
8.2.1 Poursuite de l'effort d'ingénierie et transfert en production . . . 160
8.2.2 Évolutions de l'architecture ODD/SAMURAAIE . . . 160
8.2.3 Vers une meilleure performance de l'ordonnancement du déploiement . . . 161
8.2.4 Vers une meilleure performance pour les applications déployées automatiquement . . . 162
8.2.5 Vers un système personnel d'exploitation . . . 162

Table des figures

2.1 Structure générale de l'informatique 23
2.2 Loi de Moore 25
2.3 Loi d'Amdahl 26
2.4 Architectures de ressources 27
2.5 Modèles d'applications 32

3.1 Cycle de vie d'une application sur des ressources 44
3.2 Intuition du processus de déploiement 46

4.1 Cycle permanent du déploiement 74
4.2 États possibles d'un sommet 76
4.3 Un ordinateur exécute un processus 82
4.4 Abstractions SAMURAAIE d'un ordinateur qui exécute un processus 83
4.5 Un arrêt de processus exécuté sur un ordinateur 84
4.6 Abstractions SAMURAAIE d'un arrêt d'un processus exécuté sur un ordinateur . 85
4.7 Un ajout d'un processeur à un ordinateur 86
4.8 Abstraction SAMURAAIE d'un ajout d'un processeur dans un ordinateur 87
4.9 Classes d'acteurs ODD 89
4.10 Gestionnaires ODD de déploiement 90
4.11 Exemple du traducteur d'application MPICH 93
4.12 Interactions des classes d'acteurs ODD 94
4.13 Interactions simplifiées des acteurs ODD 96
4.14 Interactions d'acteurs ODD pour la modification d'une abstraction 97
4.15 Séquences de démarrage et d'arrêt du modèle ODD 98

5.1 Hiérarchie des classes d'objets de l'implantation du modèle SAMURAAIE 102
5.2 Hiérarchie des traits des objets de l'implantation du modèle SAMURAAIE 103
5.3 Hiérarchie des classes d'objets de l'implantation du modèle ODD 104

7.1 Architecture générale de la plate-forme SALOME 132
7.2 Modèle de composants SALOME 133
7.3 Installation de la plate-forme SALOME 137
7.4 Exécution de la plate-forme SALOME 140
7.5 Interactions internes de SALOME pour l'exécution de simulations numériques . . 141
7.6 Intégration de SALOME et d'ODD/SAMURAAIE 146

Liste des tableaux

2.1 Taxonomie de Flynn 22
2.2 Exemples d'architectures d'ordinateurs classées selon la taxonomie de Flynn . . . 22

3.1 Problèmes du déploiement en informatique à haute performance 47
3.2 Étapes du déploiement supportées par les produits existants 67
3.3 Propriétés supportées par les produits existants supportant toutes les étapes du déploiement 67

4.1 Symboles des éléments et des relations SAMURAAIE 78
4.2 Noms des abstractions 78
4.3 Noms des sommets d'instance 78
4.4 Noms des sommets d'action 80
4.5 Noms des sommets d'événement 80
4.6 Double séparation des préoccupations en classes d'acteurs ODD 98

5.1 Répartition du code source 107

7.1 Correspondance entre les données de déploiement SALOME et les abstractions SAMURAAIE 147

Première partie

Introduction

Chapitre 1

Introduction

1.1 L'informatique : formidable levier du développement humain

Grâce à l'invention puis aux améliorations des ordinateurs au XX$^{\text{ième}}$ siècle, les applications du traitement automatique de l'information, c'est-à-dire les applications informatiques, se sont multipliées et diffusées dans un grand nombre d'activités humaines. Depuis les années 1950, les avancées scientifiques, techniques, et industrielles dans les domaines de l'informatique, de l'électronique, et de la télécommunication ont en effet permis la production d'ordinateurs de plus en plus puissants. La création d'un inter-réseau dans les années 1960, appelé Internet, a également permis d'envisager une infrastructure informatique mondiale dans laquelle sont actuellement réparties d'immenses quantités de données et de traitements.

La Science, la Technique, et l'Industrie emploient les applications informatiques dans leurs activités. De simples vérifications de calculs aux simulations multi-physique en passant par la gestion intégrée de groupe multinationaux, ces applications ont permis aux humains d'accélérer leur développement d'une manière exponentielle. Le secteur des technologies de l'information et de la communication est ainsi devenu un des secteurs économiques les plus fleurissants, dans lequel se sont constituées des fortunes considérables et auquel la plupart des entreprises ont recours.

Dans le secteur de l'énergie par exemple, dans un groupe tel que Électricité De France (EDF), les services de recherche, de développement, et d'ingénierie disposent d'infrastructures informatiques à haute performance afin de recourir à la simulation numérique pour leurs activités. Ces services acquièrent ou produisent eux-mêmes les applications pour ces infrastructures, applications dans les domaines variés d'un tel groupe, des courants thermiques au coeur d'une centrale nucléaire au investissements quotidiens sur les marchés financiers en passant par la prise en compte des variables météorologiques et socio-culturelles dans la détermination des niveaux de production d'électricité. L'exploitation de ces infrastructures informatiques est elle-même de plus en plus automatisée.

1.2 Des systèmes informatiques à haute performance difficiles à utiliser

Afin d'utiliser de telles infrastructures, les utilisateurs doivent déployer leurs applications sur les ressources qu'elles comportent. Dans son acception la plus large, qui est celle que ce document utilise, le déploiement d'une application est le processus qui mène celle-ci à son état de fonctionnement à partir de son état de conditionnement issu de son développement. Ce processus comporte les étapes d'allocation de ressources, d'installation de l'application sur les

ressources allouées, et d'exécution de cette application installée. Ce processus doit respecter les contraintes de compatibilité de l'application et doit tenir compte des technologies des ressources, de l'application, et des systèmes d'exploitation pour chaque étape.

Afin d'en utiliser au mieux la puissance, les applications des systèmes informatiques à haute performance héritent des propriétés des ressources de ces systèmes et donc de leur complexité. Les ressources que comportent les infrastructures informatiques à haute performance actuelles sont parallèles, dynamiques, et hétérogènes. En effet, le nombre de ressources augmente (parallélisme) et avec lui augmente la probabilité que toutes les ressources ne soient pas disponibles simultanément (dynamicité). Des ressources sont également spécialisées et fédérées afin d'en obtenir de plus puissantes encore (hétérogénéité). Les grilles d'ordinateurs sont des exemples de telles architectures. Les applications, qui composent nécessairement avec ces propriétés, se fondent donc sur des intergiciels applicatifs également parallèles, dynamiques, et hétérogènes.

Les propriétés des ressources et des applications des systèmes informatiques à haute performance complexifient le processus de déploiement d'applications, au point que les utilisateurs rencontrent de grandes difficultés. Allouer des ressources, étape déterminante pour le bon fonctionnement de l'application, nécessite de sélectionner des ressources en fonction de leur adéquation autant pour l'étape d'exécution de l'application que pour celle de son installation. Ceci peut par exemple impliquer de consulter un système d'information à propos des ressources puis d'en réserver. Installer une application est une étape souvent fastidieuse qui peut impliquer d'en compiler le code source et de la configurer, groupe de ressources par groupes de ressources. Enfin, exécuter une application nécessite de coordonner simultanément une multitude de commandes. Quand les ressources et les applications sont parallèles et répartis, hétérogènes, et dynamiques, réussir ce processus est effectivement délicat.

1.3 Vers l'automatisation de l'utilisation des systèmes informatiques à haute performance

Automatiser l'utilisation des systèmes informatiques à haute performance devrait d'abord permettre de mieux séparer les préoccupations des humains impliqués dans leur fonctionnement. Ces humains sont les administrateurs des infrastructures, les développeurs des applications, et les utilisateurs de ces applications sur ces infrastructures.

Les administrateurs détiennent l'expertise pour les infrastructures qu'ils administrent ainsi que les privilèges nécessaires afin d'installer des applications pour tous les utilisateurs de ces infrastructures. Ils doivent actuellement prendre en compte tous les besoins particuliers des utilisateurs ou choisir de ne pas les satisfaire. Afin de répondre aux besoins de certains utilisateurs, ils doivent même dédier des parties des infrastructures qu'ils administrent à des applications particulières et donc à des utilisateurs particuliers.

Les développeurs détiennent l'expertise pour les applications qu'ils développent ainsi que les privilèges nécessaires afin de modifier le code source et le conditionnement de ces applications pour tous les utilisateurs. Ils doivent actuellement prendre en compte les spécificités des services des infrastructures auxquelles les utilisateurs ont accès ou choisir de ne pas les supporter. Afin de supporter les spécificités des services de certaines infrastructures, ils doivent même limiter les besoins des applications qu'ils développent.

Les utilisateurs détiennent l'expertise de l'utilisation des applications mais ils ne sont pas nécessairement administrateurs ni développeurs. Cependant, ils doivent actuellement prendre en compte à la fois les spécificités des services des infrastructures auxquelles ils ont accès et les besoins des applications qu'ils utilisent. Afin d'utiliser certaines applications sur certaines infrastructures, ils doivent même compiler le code source et gérer l'accès simultané à plusieurs infrastructures.

En automatisant l'utilisation des systèmes informatiques à haute performance, les administrateurs pourraient se concentrer sur les infrastructures. Ils pourraient retirer des ressources défectueuses et ajouter de nouvelles ressources sans s'inquiéter de leur homogénéité ni de l'installation d'applications sur elles. Les développeurs pourraient se concentrer sur les applications. Ils pourraient en augmenter la complexité sans s'inquiéter de leurs dépendances logicielles ni de leur installation. Enfin, les utilisateurs pourraient se concentrer sur l'utilisation des applications elles-mêmes. Ils pourraient recourir beaucoup plus facilement aux systèmes informatiques à haute performance.

Automatiser l'utilisation des systèmes informatiques à haute performance devrait ensuite permettre de mieux les exploiter. Par exemple, en recourant à des algorithmes qui tiennent compte non seulement des besoins en processeurs des applications mais également de leurs besoins en mémoire vive ou de masse, en connexion, et en ressources pour leur installation, l'ordonnancement pourrait être meilleur. Supporter les infrastructures et les applications dynamiques devrait également permettre de réduire le nombre de ressources allouées mais inutilisées ainsi que le nombre de ressources non allouées, améliorant ainsi la performance énergétique des infrastructures.

1.4 Abstraction des systèmes informatiques à haute performance pour l'automatisation du déploiement d'applications dynamiques

Avant d'automatiser l'utilisation des systèmes informatiques à haute performance, il faut d'abord les représenter. Dans un système informatique donné, des représentations existent déjà mais elles sont à la fois nombreuses, disséminées, complexes, fragmentaires, et changeantes parce qu'elles sont issues des technologies particulières que comporte ce système. La première contribution de ce document est un modèle d'abstraction de ces systèmes indépendant des technologies qu'ils comportent. Le principe de ce modèle est de construire, à l'aide de concepts simples, une représentation la plus complète possible d'un système informatique, à partir de ses représentations existantes. Cette représentation intègre non seulement les caractéristiques des composants matériels et les descriptions de tâches, mais également les services permettant d'exploiter ces composants, les applications que manipulent ces tâches, leurs topologies, et les changements prévus ou déjà survenus de tous ces éléments.

Une fois représentés les systèmes informatiques à haute performance, en automatiser l'utilisation consiste à distinguer les activités que cette utilisation implique, jusqu'à pouvoir à la fois les automatiser et les coordonner. Dans un système informatique donné, les manières d'acquérir de l'information, de choisir des ressources pour une application, et d'agir sur les systèmes d'exploitation afin de faire fonctionner cette application sont également nombreuses et complexes parce qu'elles dépendent des technologies particulières que comporte ce système. La seconde contribution de ce document est un modèle d'automatisation du déploiement d'applications sur des infrastructures, modèle séparant la logique du déploiement de celles des technologies que comportent ces applications et ces infrastructures. Le principe de ce modèle est d'organiser, à l'aide d'acteurs concurrents, les activités de prise des décisions, de maintien de la représentation, de traduction de l'information existante, et de connexion avec les technologies. Une fois les ressources allouées et son application installée et exécutée par ce modèle, l'utilisateur n'a plus qu'à l'utiliser.

Cette double contribution constitue une architecture de déploiement dynamique qui a été implantée et validée. L'implantation est une preuve de concept prenant la forme d'un moteur de déploiement. La validation de cette architecture a été accomplie dans trois cas de déploiement dont la complexité augmente. Cette architecture et ce moteur ont également été intégrés dans

SALOME, une plate-forme de simulation numérique supervisée et à base de composants logiciels, afin que celle-ci leur délègue le déploiement de ses composants.

1.5 Organisation du document

Après le présent chapitre d'introduction, le reste de ce document comporte sept chapitres divisés en trois parties « Contexte et problématique », « Contribution », et « Validation ».

La partie II page 21 présente le contexte et cerne la problématique des travaux. Elle contient les chapitres 2, « Informatique à haute performance », et 3, « Déploiement dans le contexte de l'informatique à huate performance ».

Le chapitre 2 page 21 classifie les technologies impliquées dans les systèmes informatiques à haute performance. Cette classification met en évidence les propriétés de parallélisme et répartition, d'hétérogénéité, et de dynamicité des technologies de ressources, d'applications, et de systèmes d'exploitation.

Le chapitre 3 page 43 définit le déploiement d'applications et dresse un état de l'art des travaux existants dans ce domaine. Parmi les travaux dans ce domaine, peu adressent ce processus dans sa globalité et moins encore supportent toutes les propriétés des technologies impliquées dans les systèmes informatiques à haute performance.

La partie III page 73 présente les contributions. Elle contient les chapitres 4, « Architecture ODD/SAMURAAIE de déploiement dynamique d'applications », et 5, « Prototype ANGE ».

En réponse à l'analyse des technologies, la section 4.3 page 74 du chapitre 4 propose un premier modèle : le modèle d'abstraction des systèmes informatiques à haute performance. Ce modèle est appelé modèle d'abstraction système pour les actions, les instances, et les événements de l'utilisateur, des ressources, et des applications (SAMURAAIE). Il permet d'abstraire les contraintes architecturales et logistiques régissant le fonctionnement des systèmes informatiques, en utilisant le concept de correspondance entre contenants et contenus.

En réponse cette fois à l'analyse des travaux existants dans le domaine, la section 4.4 page 88 du chapitre 4 propose un second modèle : le modèle d'automatisation du déploiement d'applications. Ce modèle est appelé modèle de déploiement sur demande (ODD). Afin de traiter continuellement les trois étapes du déploiement, ce modèle organise quatre classes d'acteurs que sont les gestionnaires, les abstracteurs, les traducteurs, et les connecteurs.

Le chapitre 5 page 101 propose une implantation, sous la forme d'un moteur extensible de déploiement, de l'architecture de déploiement dynamique ODD/SAMURAAIE obtenue de la combinaison des deux modèles proposés. Cette implantation est appelée Adage Nouvelle Génération (ANGE), reprenant ainsi le nom d'un outil de déploiement issu de travaux précédents.

La partie IV valide l'architecture et le moteur proposés en partie III. Elle contient les chapitres 6, « Automatisation du déploiement d'applications parallèles, paramétriques, et en flot de travail », et 7, « Automatisation du déploiement de simulations numériques SALOME ».

Le chapitre 6 page 111 valide la combinaison des deux modèles proposés en étudiant l'automatisation, avec l'architecture ODD/SAMURAAIE, de trois cas de déploiement. Il s'agit de déployer une application parallèle sur un ordinateur, une application paramétrique sur une grappe d'ordinateurs, et une application en flot de travail sur une grille d'ordinateurs.

Le chapitre 7 page 7 valide la combinaison des deux modèles et l'implantation proposés en automatisant, avec l'architecture ODD/SAMURAAIE et le moteur ANGE, le déploiement de simulations numériques SALOME.

Enfin, le chapitre 8 page 159 présente la conclusion et donne des perspectives pour ces travaux.

Deuxième partie

Contexte et problématique

Chapitre 2

Informatique à haute performance

2.1 Introduction

Du calcul mental de la monnaie à rendre, par un commerçant, à la simulation numérique de phénomènes naturels et artificiels, par des ingénieurs, la manipulation des nombres est une activité avec laquelle les humains ont une longue expérience. De nombreuses techniques ont progressivement vu le jour, des abaques grecques au système arithmétique positionnel arabe, en passant par la règle à calcul ou la pascaline. Les notations mêmes des nombres ont évolué avec ces techniques, de simples juxtapositions de bâtons aux nombres arabo-européens, en passant par les nombres romains. Les cailloux avec lesquels les humains comptaient ont donné leur racine latine à cette activité : calculer.

Aujourd'hui, les humains apprennent toujours à calculer de tête et avec des outils rudimentaires pour « poser » leurs opérations. Ils font cependant la majeure partie de leurs calculs avec des technologies qui ont vu le jour lors de la formidable accélération du progrès technique, au XX$^{\text{ième}}$ siècle. L'automatisation et la haute performance des calculs sont désormais fondées sur l'électronique et l'informatique, et sont exploitées pour la recherche scientifique ainsi que pour d'autres activités humaines, telles que les affaires.

Ce chapitre propose une classification des technologies informatiques dans le domaine du calcul à haute performance. Il commence par en donner les principes. Il présente ensuite les trois classes de technologies complémentaires que sont les ressources, les applications, et les systèmes d'exploitation. Enfin, il discute les éléments mis en évidence par cette classification.

2.2 Principes

Dans le dictionnaire gratuit en ligne de l'informatique (*Free On-Line Dictionary Of Computing*, FOLDOC) [1], la définition d'un ordinateur est la suivante :

> "Un ordinateur est une machine programmable sachant manipuler des symboles. Les ordinateurs peuvent effectuer des procédures complexes et répétitives de manière rapide, précise, et fiable et peuvent mémoriser et retrouver rapidement une grande quantité de données.
>
> Les composants physiques avec lesquels un ordinateur est construit (des circuits électroniques et des dispositifs d'entrée-sortie) sont appelés "matériel" (*hardware*). La plupart des ordinateurs ont quatre types de composants matériels : unité centrale de traitement (/Central Processing Unit/, CPU), entrée, sortie, et mémoire. L'unité centrale de traitement exécute des programmes («logiciel», /software/) qui indiquent quoi faire à l'ordinateur. Les dispositifs d'entrée et de sortie (E-S) permettent à

	Instruction simple	Instruction multiple
Flux de données simple	SISD	MISD
Flux de données multiple	SIMD	MIMD

Table 2.1 – Taxonomie de Flynn

Classe	Exemple
SISD	Ordinateurs strictement mono-processeur
SIMD	Processeurs vectoriels
MISD	Ordinateurs tolérant les défaillances
MIMD	Ordinateurs multi-processeur

Table 2.2 – Exemples d'architectures d'ordinateurs classées selon la taxonomie de Flynn

> l'ordinateur de communiquer avec l'utilisateur et le monde extérieur. Il y a plusieurs sortes de mémoires : la mémoire rapide, onéreuse, et à court terme (la RAM par exemple) pour mémoriser des résultats intermédiaires, et la mémoire plus lente, plus économique, et à long terme (disque et cassette magnétiques par exemple) pour mémoriser les programmes et les données entre les tâches."

Les ordinateurs ont des mémoires, des processeurs, et des dispositifs faisant entrer et sortir les données mémorisées. Cependant, à la fin des années 1940, la notion de programme a remarquablement évolué. Auparavant, les programmes étaient câblés matériellement dans l'ordinateur, ils faisaient partie du processeur lui-même. Maintenant en revanche, ainsi que le suggère la définition ci-dessus, les ordinateurs mémorisent les programmes comme des données, ce qui facilite beaucoup leur exploitation. Cette évolution est appelée architecture d'ordinateur à programme mémorisé, ou architecture de Von Neumann, du nom de son inventeur.

En 1966, Flynn a proposé une taxonomie des architectures d'ordinateurs [2], voir le tableau 2.1 page 22. Cette taxonomie a deux dimensions : instructions («I») d'une part et flux de données («D») d'autre part. Chaque dimension n'a que deux valeurs possibles : « simple » («S») ou bien « multiple » («M»), c'est-à-dire respectivement non-concurrence ou bien concurrence. Par combinaison, cette taxonomie comporte donc quatre classes d'architectures dont le tableau 2.2 page 22 donne des exemples. La classe la plus répandue aujourd'hui est l'architecture multi-instruction multi-flux de données (MIMD). Ces architectures sont tellement répandues qu'une subdivision est fréquemment utilisée. Parmi elles, on peut effectivement distinguer celles qui exécutent un seul programme («P») ou bien plusieurs programmes à la fois (architectures SPMD ou bien MPMD).

2.2.1 Structure générale de l'informatique

Le premier principe de la classification que propose ce chapitre est de généraliser à l'informatique la structure d'un ordinateur, définie ci-dessus, en séparant ordinateur et programme, d'une part, et mémoires et données, processeurs et traitements, et dispositifs d'entrée-sortie et flux, d'autre part. Le traitement de l'information peut être automatisé grâce à une infrastructure informatique programmable, c'est-à-dire un ou plusieurs ordinateurs interconnectés par des dispositifs d'entrée-sortie supplémentaires. Une infrastructure informatique comporte à la fois des éléments matériels et logiciels. La figure 2.1 page 23 illustre ce premier principe de classification.

Les éléments matériels, tels que des mémoires, des processeurs, et des dispositifs d'entrée-sortie, sont appelés ressources informatiques ou ressources. Les éléments logiciels sont appelés systèmes d'exploitation et permettent d'exécuter des programmes traitant des données. Ces programmes sont en effet associés à des données, comportent des traitements, et accèdent aux

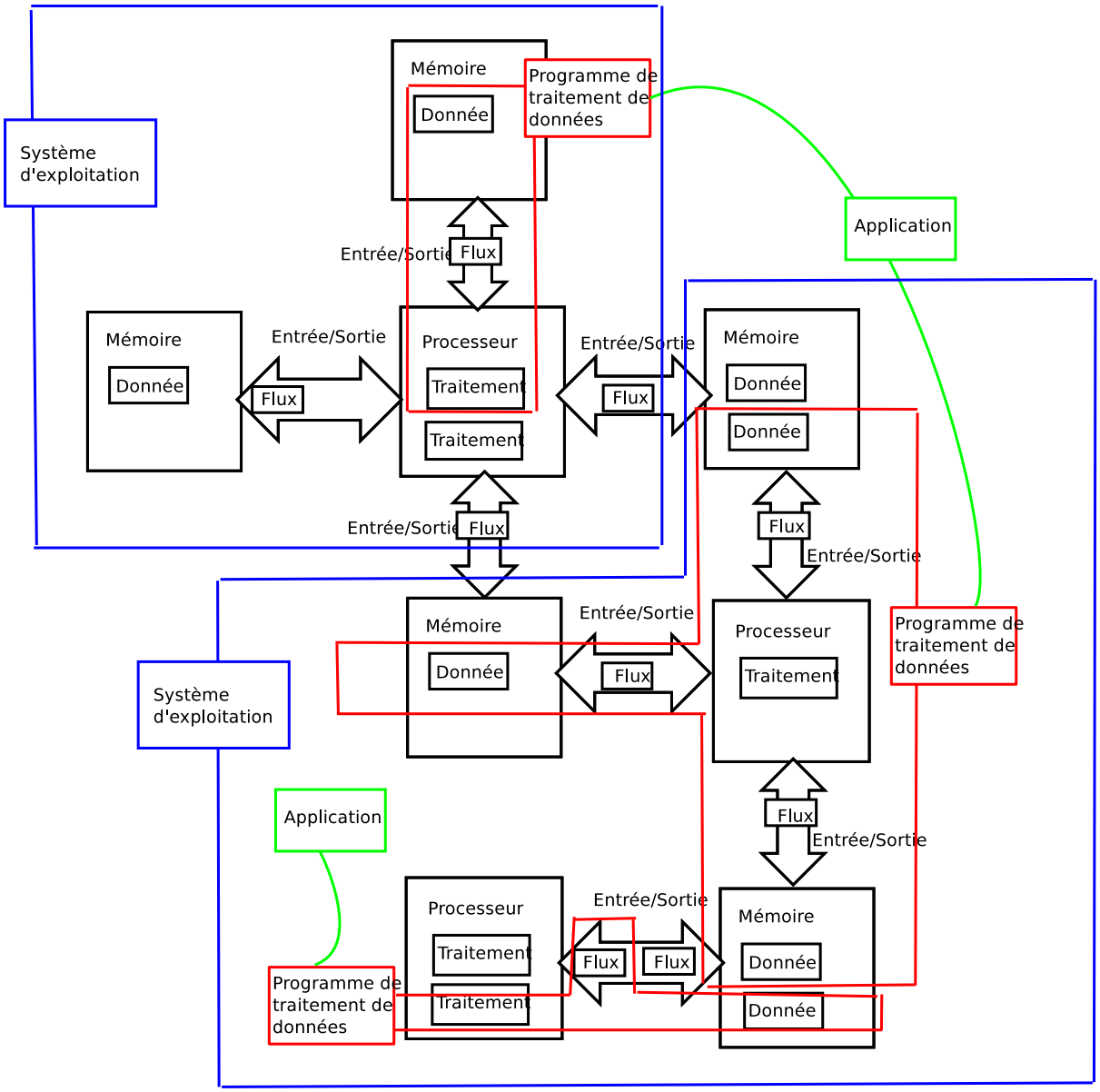


FIGURE 2.1 – Structure générale de l'informatique

données. Les données sont l'information mémorisée par les mémoires, les traitements sont les procédures effectuées par les processeurs, et les flux sont les données entrant et sortant par les dispositifs d'entrée-sortie. Cet ensemble ainsi formé, qui utilise des ressources, est appelé application. Ces définitions, qui sont pour la plupart récursives, structurent la classification des technologies d'informatique à haute performance.

L'informatique est « à haute performance » quand sa qualité d'exécution est exceptionnelle. Une manière de qualifier l'exécution est de mesurer des quantités. Les premiers quantificateurs sont les caractéristiques des éléments matériels impliqués : les tailles des mémoires en octet, les fréquences d'horloge des processeurs en Hertz, et les débits et latences des dispositifs d'entrée-sortie entre les mémoires et les processeurs respectivement en octet par seconde et en seconde. Ces caractéristiques, indépendantes, ont rapidement été remplacées par des quantificateurs qui les intègrent. Ces quantificateurs mesurent des vitesses, c'est-à-dire des rapports entre des nombres d'opérations et le temps (*Instructions Per Second*, IPS, et *FLoating point Operation Per Second*, FLOPS).

La vitesse d'exécution de l'informatique peut être estimée à partir des caractéristiques matérielles de l'infrastructure, ou mesurée en exécutant une application développée spécifiquement (*benchmarking*). On obtient ainsi des valeurs dites respectivement « théoriques » ou « soutenues ». Par exemple, l'application permettant de classer les 500 ordinateurs ayant les meilleures performances (vitesse d'exécution soutenue en FLOPS) au monde [3] est LINPACK [4].

Depuis quelques années, les auteurs de LINPACK proposent de compléter les mesures avec six autres applications : cette suite d'applications est appelée le défi de l'informatique à haute performance (*High-Performance Computing Challenge*, HPCC) [5]. D'autres suites existent, telles que celles produites par la corporation d'évaluation standard de performance (*Standard Performance Evaluation Corporation*, SPEC) et la suite parallèle NAS (*NAS Parallel Benchmark*, NPB) [6] produite par l'administration nationale étasunienne pour l'aéronautique et l'espace (*National Aeronautics and Space Administration*, NASA).

Les quantificateurs de vitesse d'exécution ont cependant des limites. Des différences sensibles peuvent apparaître entre les vitesses d'exécution théoriques, celles mesurées par des applications spécifiques, et celles constatées pour les applications réelles. D'autres différences peuvent apparaître entre les vitesses d'exécution à différents régimes d'exploitation, des applications simples pouvant paradoxalement s'exécuter plus lentement que des applications complexes. Des paramètres autres que le nombre d'opérations et le temps permettraient de mieux qualifier l'informatique.

On pourrait par exemple rapporter la vitesse d'exécution d'une infrastructure informatique au coût de son matériel, à la surface qu'elle occupe, ou à l'énergie qu'elle consomme. On pourrait également rapporter la vitesse constatée à celles soutenue et théorique. Née en novembre 2007, la liste des 500 infrastructures informatiques les plus économes en énergie (*Green500*) [7] encourage les acteurs de l'informatique à haute performance à prendre en compte la durabilité de leurs infrastructures. Le projet EcoGrappe va également dans ce sens. S'ils peuvent être étudiés, les coûts, les surfaces, l'énergie consommée, et le nombre d'opérations perdues sont toutefois encore peu pris en compte dans l'élaboration du classement principal.

2.2.2 Facteurs d'amélioration des performances

Le second principe de la classification que propose ce chapitre est de distinguer trois facteurs améliorant l'informatique. Outre l'amélioration individuelle, ces facteurs sont la multiplication, la spécialisation, et la variation du nombre d'éléments impliqués. L'amélioration individuelle des éléments matériels suit la loi (ou prophétie auto-réalisatrice ?) de Moore [8], illustrée par la figure 2.2 page 25, loi qui stipule que le nombre de transistors dans les ressources doublerait tous les

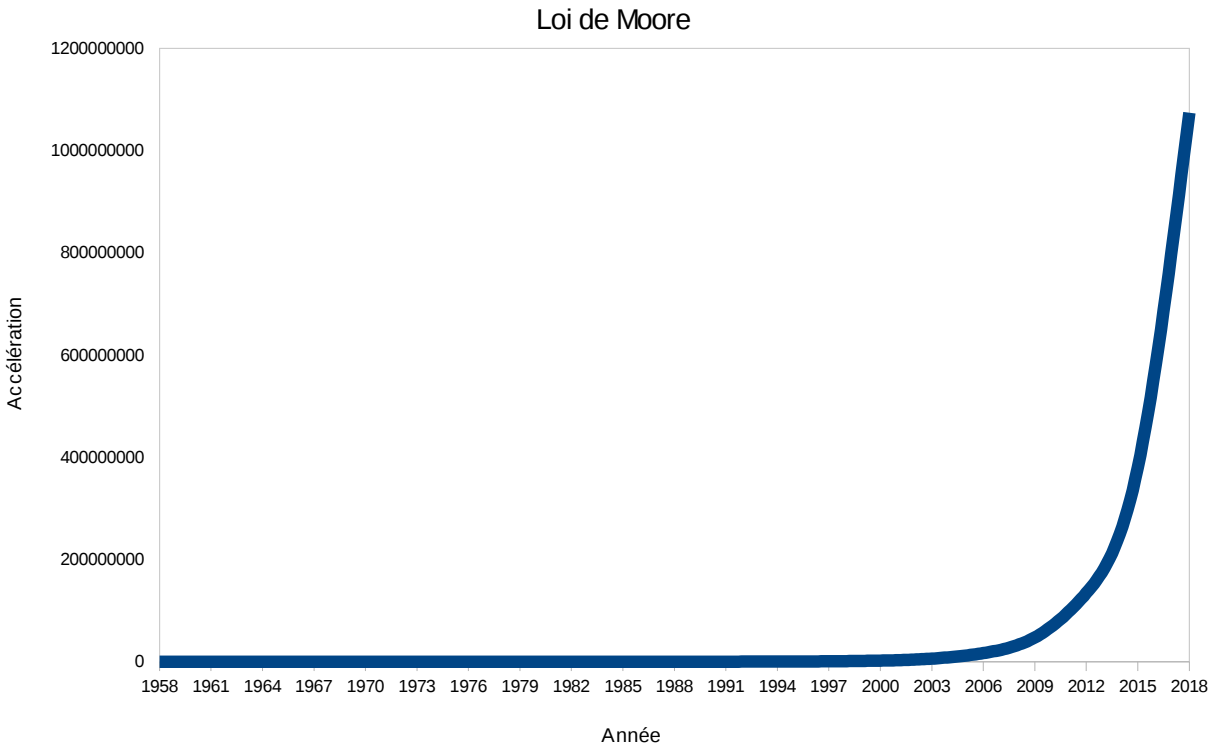


FIGURE 2.2 – Loi de Moore

deux ans. Pour les éléments logiciels, cette amélioration correspondrait à l'optimisation de leur programmation. Les définitions des trois facteurs d'amélioration sont les suivantes :

Paralléliser Multiplier des éléments homogènes, matériels et logiciels, c'est-à-dire paralléliser les traitements, permet d'augmenter le nombre d'opérations en temps constant. La loi d'Amdahl [9], illustrée par la figure 2.3 page 26, stipule toutefois que le nombre de traitements ne pouvant être exécutés qu'en séquence n'est jamais nul, et qu'il limite cette accélération.

Spécialiser Différencier des éléments initialement homogènes, matériels et logiciels, c'est-à-dire les rendre hétérogènes, permet de diminuer le nombre d'opérations requises, d'augmenter le nombre d'opérations possibles, ou les deux afin d'exécuter une partie des traitements en moins de temps qu'il n'en aurait fallu si les éléments étaient restés homogènes.

Varier le nombre Augmenter ou diminuer le nombre d'éléments, matériels et logiciels, homogènes ou hétérogènes, permet de majorer le rapport du nombre d'opérations effectuées à celui d'opérations possibles, ce qui permet d'optimiser l'exploitation des ressources.

La suite de ce chapitre illustre l'articulation des deux principes présentés ci-dessus avec des exemples d'architectures, d'une part, et le détail de leurs éléments, d'autre part.

2.3 Ressources, applications, et systèmes d'exploitation

2.3.1 Ressources

Les ressources sont les éléments matériels d'une infrastructure informatique. Cette sous-section présente les quatre types d'architectures de ressources que sont l'ordinateur, la grappe, la grille, et le nuage d'ordinateurs. Puis elle présente les trois types d'éléments matériels que sont les mémoires, les processeurs, et les dispositifs d'entrée-sortie également appelés connecteurs.

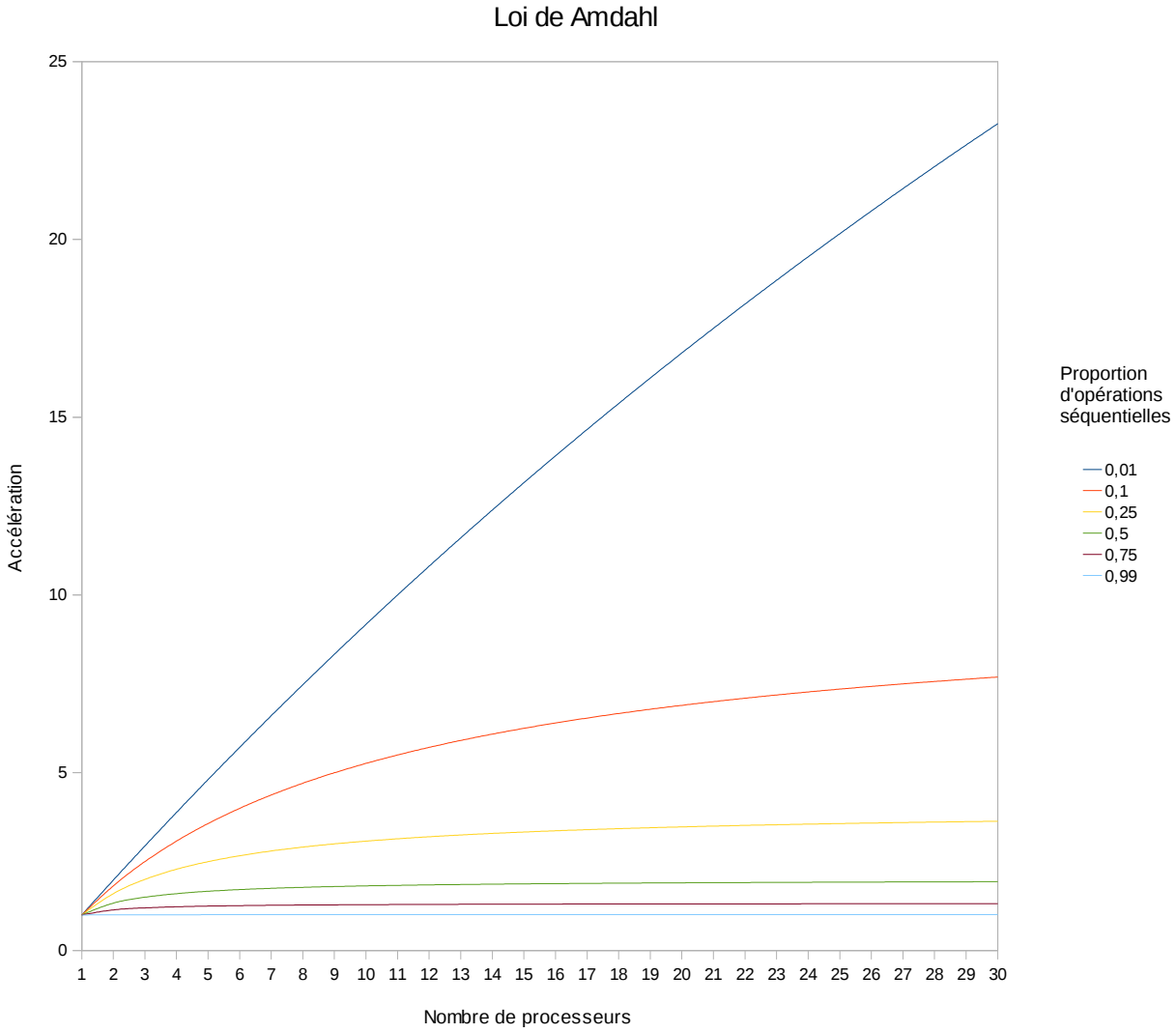


FIGURE 2.3 – Loi d'Amdahl

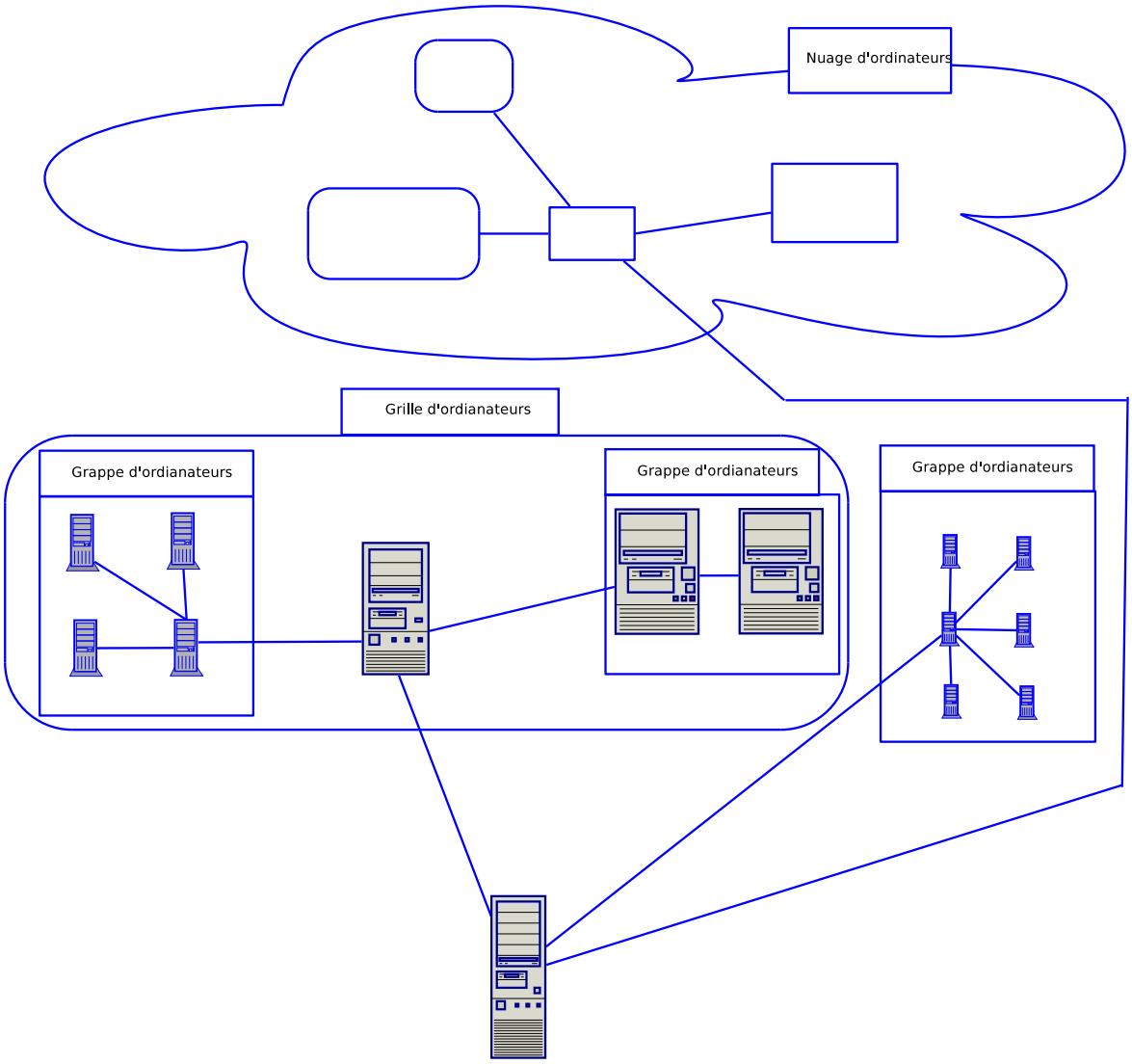


FIGURE 2.4 – Architectures de ressources

Architectures de ressources

La figure 2.4 page 27 illustre les architectures d'ordinateurs, de grappes, de grilles, et de nuages d'ordinateurs.

Ordinateurs Les ressources d'un ordinateur ont deux caractéristiques que ne précise pas la définition de FOLDOC d'un ordinateur, citée page 21. Par rapport aux autres architectures de ressources, celle d'un ordinateur est à la fois électriquement indépendante et indivisible. En d'autres termes, les ressources d'un ordinateur comportent effectivement des mémoires, des processeurs, et des connecteurs, mais qu'elles aient leur propre alimentation électrique et qu'aucune de leurs parties ne puisse fonctionner séparément les distinguent. La seconde caractéristique est reliée à cette indépendance électrique. Les ressources d'un ordinateur comportent effectivement une mémoire à accès lent, dite « de masse », dans laquelle mémoriser des données entre des exécutions de tâches, mais cette mémoire permet également aux données de persister quand les ressources ne sont plus alimentées électriquement, c'est-à-dire quand l'ordinateur est éteint ou hors service.

Le constructeur Hewlett-Packard produit par exemple des ordinateurs personnels (*Personal Computers*, PCs) de bureau XW6600 comportant des ressources, telles que les suivantes :

- Une mémoire SDRAM à accès rapide, dite « vive », de 8 Go et une mémoire de masse de 300 Go sur deux disques durs ;
- Dix cœurs de processeurs dont ceux de deux processeurs quadri-cœur Intel Xeon 2,33 GHz généraux, celui d'un processeur graphique NVIDIA Quadro, et celui d'un processeur Broadcom Gigabit Ethernet pour connecter l'ordinateur en réseau ;
- Des connecteurs intégrés DDR pour les accès des processeurs généraux à la mémoire vive, SATA pour la mémoire de masse, et PCI Express pour les processeurs spécifiques.

Un ordinateur disposant de telles ressources n'a cependant aucune chance de figurer parmi les 500 infrastructures informatiques les plus performantes du monde.

Les ressources des super-ordinateurs RoadRunner et Blue Gene d'International Business Machines (IBM), Jaguar de Cray Incorporated, et Pleiades de Silicon Graphics Incorporated (SGI) en font les cinq infrastructures informatiques (Blue Gene étant représenté deux fois) les plus performantes du monde, dont deux soutiennent des performances au-delà de 1 PFLOPS (10^{15} FLOPS).

RoadRunner Les ressources de RoadRunner comporte presque 6500 processeurs bi-cœur AMD Opteron interconnectés par Infiniband, un processeur PowerXCell 8i étant adjoint à chaque cœur de processeur.

Jaguar Les ressources de Jaguar sont constituées d'unités Cray XT4 et XT5, le tout comportant plus de 180000 cœurs de processeur AMD Opteron interconnectés par SeaStar ou par Infiniband. La mémoire de masse partagée entre les processeurs est un autre produit Cray appelé Spider.

Pleiades Les ressources de Pleiades sont constituées d'unités SGI Altix. Cette gamme propose des super-ordinateurs d'architecture NUMAflex comportant jusqu'à 1024 cœurs de processeur Intel Xeon et 128 To (1024 Go) de mémoire vive, répartie et partagée, cœurs interconnectés par NUMAlink.

Blue Gene Dans la version la plus performante (Blue Gene/P), Blue Gene est constituée d'armoires pouvant contenir 32 lames, chaque lame pouvant contenir 16 nœuds, chaque nœud pouvant comporter deux processeurs PowerPC 440 cadencés à 700 MHz et de 1 Go de mémoire vive. Ces nœuds sont interconnectés dans trois réseaux : un tridimensionnel pour les communications nœud à nœud, un pour les communications collectives, et un pour les interruptions globales. Par ailleurs les lames sont connectées vers l'extérieur par un réseau Ethernet d'administration et les nœuds se partagent une mémoire de masse externe.

Lorsqu'un ordinateur comporte plusieurs processeurs, soit ceux-ci partagent une même mémoire vive, soit plusieurs mémoires vives coexistent. Dans le premier cas, on parle de multi-processeur symétrique (*Symetric Multi-Processor*, SMP). Dans le second, on parle de mémoire répartie et, le temps d'accès d'un processeur à une zone mémoire dépendant de leur proximité, d'accès non uniforme à la mémoire (*Non-Uniform Memory Access*, NUMA). Des problèmes de cohérence entre les mémoires vives des processeurs pourraient d'ailleurs apparaître si plusieurs d'entre eux accédaient simultanément à une même zone mémoire. Les contrôleurs de mémoire d'un tel ordinateur peuvent organiser cette cohérence, on parle alors d'accès non uniforme à la mémoire mais cohérent (l'accès) en cache (*cache-coherent Non-Uniform Memory Access*, ccNUMA). Les super-ordinateurs Altix de SGI, par exemple, ont une architecture ccNUMA.

Grappes d'ordinateurs Les ressources d'une grappe d'ordinateurs comportent les ressources de plusieurs ordinateurs homogènes interconnectés. La performance des accès des processeurs

aux mémoires vives est différente selon que ces ressources soient localisées dans un même ordinateur. Le nombre d'ordinateurs peut varier au cours du temps, particulièrement pour des raisons de maintenance. Autrement dit, les ressources d'une grappe d'ordinateurs sont parallèles et dynamiques. Plus économiques, des grappes d'ordinateurs personnels sont fréquemment utilisées en remplacement de super-ordinateurs. Quand elles comportent un grand nombre d'ordinateurs, les grappes rivalisent avec les super-ordinateurs dans les classements mondiaux des infrastructures informatiques les plus performantes.

Apparue à la NASA en 1994, l'architecture de grappe d'ordinateurs appelée Beowulf [10] est fréquemment mise en œuvre. Constituée de matériel ordinaire et donc économique, cette architecture distingue deux types d'ordinateurs : le nœud passerelle, ou encore frontal, et les nœuds de calcul. Le nœud passerelle comporte deux connecteurs réseau, le premier vers l'extérieur, le second vers le sous-réseau dans lequel sont isolés les nœuds de calcul. Il offre une mémoire de masse que les nœuds de calcul se partagent. Le nœud passerelle et les nœuds de calcul sont interconnectés dans le sous-réseau. Cette architecture a fait l'objet de nombreuses variantes. Par exemple, le nœud frontal est souvent plus performant que les nœuds de calcul pris séparément, le réseau des nœuds de calcul n'est pas toujours isolé, plusieurs nœuds frontaux peuvent se répartir leur charge de travail, et la mémoire de masse partagée est parfois offerte par un nœud supplémentaire. Cependant le nom de l'architecture persiste.

Grilles d'ordinateurs Les ressources d'une grille d'ordinateurs comportent les ressources de plusieurs ordinateurs, plusieurs grappes d'ordinateurs, ou les deux [11]. Quand elle ne comporte que des grappes, une grille est parfois appelée fédération de grappes voire grappe de grappes. Les interconnexions des grappes peuvent être beaucoup plus longues que celles entre les ordinateurs d'une même grappe, un niveau supplémentaire apparaît donc dans la hiérarchie de l'infrastructure informatique. De même que dans une grappe, le nombre d'ordinateurs et de grappes d'une grille peut varier au cours du temps mais, contrairement à une grappe, ces ordinateurs peuvent être différents. En d'autres termes, les ressources d'une grille sont parallèles, hétérogènes, et dynamiques.

Plusieurs grilles d'ordinateurs sont connues dans le monde. Le projet de rendre les grilles aptes pour la science numérique (*Enabling Grids for E-SciencE*, EGEE) est soutenu par la commission européenne et par de nombreux partenaires académiques et privés. Son objectif est de produire une infrastructure informatique fiable et partagée entre une cinquantaine de pays. Elle comporte 260 centres de ressources informatiques, répartis de par le monde, et fédère ainsi 92000 cœurs de processeurs et plusieurs petaoctets de mémorisation que peuvent utiliser plusieurs milliers de scientifiques.

Nuages d'ordinateurs Les ressources d'un nuage d'ordinateurs sont fournies sur demande. En effet, le principe du nuage est de délocaliser les ressources chez un fournisseur et de continuer de les utiliser, par Internet, en ne les possédant plus. Le pictogramme représentant Internet est souvent un nuage, il a donc donné son nom à ce principe architectural. Les architectures de ressources ainsi fournies peuvent donc théoriquement être constituées d'ordinateurs, de grappes, ou de grilles. Cependant, un fournisseur devant honorer des demandes pour un usage immédiat, il ne fournit généralement pas des ordinateurs réels mais plutôt des machines virtuelles. Cette architecture étant récente, il existe encore peu d'offre en informatique à haute performance. Parmi ses services de toile, l'entreprise Amazon propose par exemple un nuage d'ordinateurs virtuels appelé nuage élastique de calcul (*Elastic Compute Cloud*, EC2).

Mémoires, processeurs, et connecteurs

Mémoires Les mémoires sont les ressources emmagasinant l'information. Leurs principales caractéristiques sont la persistence et la taille en octet de cette information. Dans l'ordre croissant de persistance figurent les mémoires suivantes :

- Registres et mémoires cache dans les processeurs ;
- Mémoire vive à accès aléatoire, dynamique, et synchrone (*Synchronous Dynamic Random Access Memory*, SDRAM) dans les ordinateurs ;
- Mémoires de masse, telles que les disques durs (*Hard Disk Drives*, HDDs) et les disques à états solides (*Solid State Disks*, SSDs) des ordinateurs ;
- Baies de stockage dans les super-ordinateurs, les grappes, et les grilles d'ordinateurs ;
- Mémoires mortes, telles que les mémoires en lecture seule (*Read Only Memories*, ROMs) dont les mémoires électriquement effaçables et programmables (*Electrically Erasable and Programmable Read Only Memories*, EEPROMs), dans les ordinateurs.

En ce qui concerne les tailles, on constate que plus les mémoires sont matériellement éloignées des processeurs, plus leurs tailles sont grandes. Physiquement, la mémorisation repose sur des techniques électro-magnétiques, optiques, ou les deux. Par ailleurs, afin de les renforcer, les mémoires peuvent comporter des contrôleurs électromécaniques, des codes de correction d'erreur (*Error Correction Codes*, ECCs), voire être redondantes. Pour la mémoire de masse, les techniques des baies redondantes de disques durs indépendants (*Redundant Arrays of Independent Disks*, RAID) sont fréquemment mises en œuvre. Des constructeurs de mémoires sont par exemple Kingston, Seagate, et Hewlett Packard.

Processeurs Les processeurs sont les ressources qui effectuent les traitements sur l'information. Leurs principales caractéristiques sont l'architecture de jeu d'instructions (*Instruction Set Architecture*, ISA), la fréquence en Hertz de l'horloge, et la capacité d'adressage en bits. La miniaturisation des processeurs permet aujourd'hui d'en faire tenir plusieurs là où il n'en tenait qu'un, on parle alors de processeurs « multi-cœur ». Les cinq catégories suivantes de processeurs peuvent être distinguées :

- Processeurs généraux (*Central Processing Units*, CPUs), tels que le processeur Intel I7 de la famille de processeurs Nehalem ;
- Processeurs graphiques (*Graphical Processing Unit*, GPUs), tels que le ATI Radeon HD 4770 d'Advanced Micro Devices (AMD) ou des produits concurrents, de NVIDIA ;
- Processeurs de connecteur, tels que ceux produits par 3Com ou Quadrics ;
- Processeurs hétérogènes, tels que le processeur IBM Cell ;
- Circuits intégrés programmables (*Field-Programmable Gate Arrays*, FPGAs).

Connecteurs Les connecteurs sont les ressources qui permettent les accès des processeurs à l'information mémorisée. En connectant des mémoires et des processeurs, ils déterminent la topologie des ressources. Leurs principales caractéristiques sont le débit en octets par seconde, leur latence en seconde, et le protocole d'accès supporté. Les types de connecteurs sont les suivants :

- Connecteurs point à point de processeurs, tels que l'HyperTransport et le QuickPath Interconnect d'Intel ;
- Connecteurs entre processeurs et mémoires, tels que le circuit Northbridge ou le superconcentrateur (*super-hub*, shub) de SGI entre processeur général et mémoire vive, l'attachement en série de technologie avancée (*Serial Advanced Technology Attachment*, SATA) et l'interface de petit système d'ordinateur (*Small Computer System Interface*, SCSI) entre processeur général mémoire de masse, et l'accès direct à la mémoire (*Direct Memory Access*, DMA) entre processeurs spécifiques et mémoires vives ;

– Connecteurs d'ordinateurs en réseaux locaux (*Local Area Networks*, LANs), tels que Ethernet, Myrinet, NUMAlink, et l'accès direct à la mémoire à distance (*Remote Direct Memory Access*, RDMA) dont Fiber Channel et Infiniband.
– Connecteurs d'ordinateurs en réseaux à longue distance (*Wide Area Networks*, WANs), tels que le mode de transfert asynchrone (*Asynchronous Transfer Mode*, ATM), la commutation de label de protocole multiple (*Multi-Protocol Label Switching*, MPLS), et le relais de cadres (*Frame Relay*).

Afin de faire circuler l'information, les connecteurs utilisent physiquement des dispositifs électroniques, optiques, ou les deux. Les connecteurs comportent généralement des adaptateurs et des câbles reliant ces adaptateurs. Les adaptateurs sont généralement des cartes d'interface élaborées comportant des mémoires vives et processeurs spécialisés. Sur leur chemin, les câbles en cuivre ou bien en fibre optique peuvent rencontrer des répétiteurs, des commutateurs, des routeurs, des pare-feux, etc. Les mémoires et les processeurs ont des adaptateurs. Outre les constructeurs de mémoires et de processeurs, des constructeurs de connecteurs sont par exemple Cisco, Myricom, et Mellanox.

Discussion

Distinguer un super-ordinateur d'une grappe d'ordinateurs est parfois difficile. En effet, les super-ordinateurs et les ordinateurs comportent toujours plus de processeurs et des processeurs comparables sinon similaires. Blue Gene par exemple comporte même des processeurs moins performants que ceux d'un ordinateur de bureau. Par ailleurs l'indépendance électrique des sous-parties des super-ordinateurs se rapproche fortement de celle des ordinateurs d'une grappe. L'intégration des ressources dans des armoires offre les mêmes possibilités électriques. En revanche, les super-ordinateurs, tels que Pleiades ou Blue Gene, gardent une avance très nette sur les grappes d'ordinateurs par rapport à la performance de leur connecteurs, et donc du rapport des performances des communications inter-traitement à celles des traitements.

Le nombre en constante augmentation des ressources les rend de plus en plus parallèles et dynamiques. On constate également un intérêt croissant pour des processeurs spécialisés, tels que les processeurs graphiques, rendant les ressources de plus en plus hétérogènes. Les trois facteurs d'amélioration des performances coexistent donc dans les ressources des infrastructures informatiques actuelles.

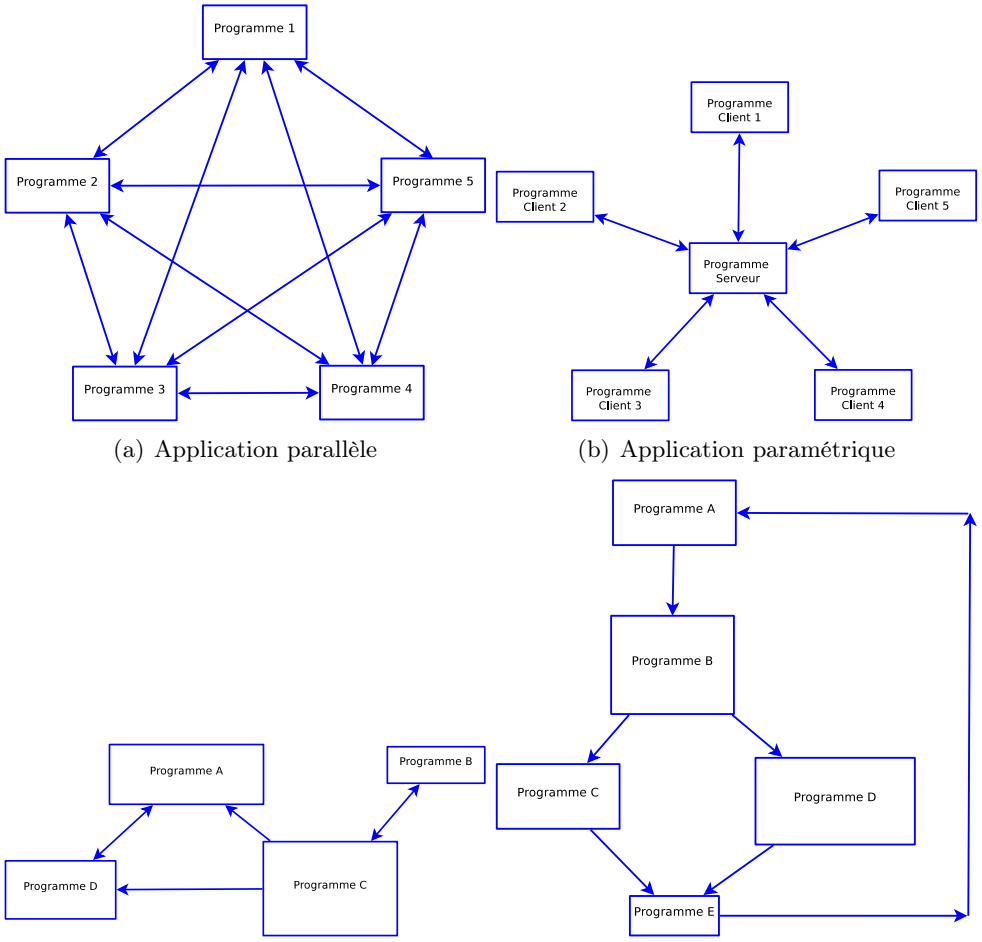
2.3.2 Applications

Les applications sont les éléments logiciels programmés afin d'être exécutés sur les ressources. Cette sous-section présente cinq types de modèles d'applications : les applications paramétriques, parallèles, réparties, en flots de travail, et à base de composants. Puis elle présente les trois types d'éléments logiciels des applications que sont les données, les traitements, et les flux.

Modèles d'applications

Les figures 2.5(a), 2.5(b), 2.5(c), et 2.5(d) page 32 illustrent respectivement les modèles d'applications parallèles, paramétriques, réparties, et en flot de travail.

Applications parallèles Une application parallèle comporte un seul programme s'exécutant simultanément sur plusieurs processeurs pendant une partie de la durée de l'exécution de l'application. Ce programme lit les données d'entrée, les traite, et écrit les données de sortie. Ses exécutions simultanées se partagent ces activités et peuvent intercommuniquer pour ce faire. On



(a) Application parallèle

(b) Application paramétrique

(c) Application répartie

(d) Application en flot de travail

FIGURE 2.5 – Modèles d'applications

parle également d'applications mono-programme multi-flux de données (*Single Program Multiple Data*, SPMD).

Une application parallèle peut être dynamique. En effet, si le nombre de processeurs requis simultanément varie pendant la durée de l'exécution de l'application, alors cette application est dynamique. Parmi les technologies qui permettent de programmer des applications parallèles figurent par exemple les blocs de construction de fils d'exécution (*Threading Building Blocks*, TBB) [12], les fils d'exécution POSIX, la bibliothèque de fils d'exécution POSIX natifs (*Native POSIX Threads Library*, NPTL), le langage de programmation Fortran de haute performance (*High-Performance Fortran*, HPF) [13], le standard de multi-traitement ouvert (*Open Multi-Processing*, OpenMP) [14], et l'interface de passage de messages (*Message Passing Interface*, MPI) [15]. La plupart de ces technologies permettent de programmer des applications non seulement parallèles mais également dynamiques.

Applications paramétriques Une application paramétrique comporte généralement deux parties. La première est un programme, appelé serveur, maître, ou lanceur. Le serveur lit les données d'entrée, écrit les données de sortie, et est exécuté pendant toute la durée de l'exécution de l'application. La seconde partie est un ou plusieurs programmes appelés clients, esclaves, ou codes paramétriques. Les clients sont exécutés plusieurs fois par application, en série, en parallèle, ou les deux. Pendant leurs exécutions, les clients communiquent avec le serveur afin de recevoir des données à traiter, les traitent, puis communiquent à nouveau avec le serveur afin de renvoyer les données traitées. Les clients n'intercommuniquent pas.

Une application paramétrique peut être à la fois parallèle, hétérogène, et dynamique. En effet, si un client est exécuté plusieurs fois simultanément, alors l'application est parallèle. Si plusieurs clients sont exécutés sur des processeurs hétérogènes, tels qu'un processeur général et un processeur graphique, alors l'application est hétérogène. Enfin, si le nombre de clients exécutés simultanément varie pendant la durée de l'exécution de l'application, alors cette application est également dynamique. La famille d'applications « @home », telles que SETI@home [16], fondées sur l'infrastructure ouverte de Berkeley pour l'informatique en réseau (*Berkeley Open Infrastructure for Network Computing*, BOINC) [17] sont des exemples d'applications paramétriques.

Applications réparties Une application répartie comporte plusieurs programmes intercommuniquant et s'exécutant simultanément sur plusieurs processeurs. Les programmes se partagent les activités de lecture des données d'entrée, de traitement de ces données, et d'écriture des données de sortie. On parle également d'applications multi-programme multi-flux de données (*Multiple Program Multiple Data*, MPMD).

Une application répartie peut être à la fois hétérogène et dynamique. En effet, si plusieurs programmes sont exécutés sur des processeurs hétérogènes, alors l'application est hétérogène. Si le nombre de programmes exécutés simultanément varie pendant la durée de l'exécution de l'application, alors cette application est également dynamique. Parmi les technologies permettant de programmer des applications réparties figurent par exemple les appels de procédure à distance (*Remote Procedure Call*, RPC), tels que la boîte à outils d'ingénierie répartie et interactive (*Distributed Interactive Engineering Toolbox*, DIET) [18], l'architecture commune de courtier en requêtes d'objets (*Common Object Request Broker Architecture*, CORBA) [19] du groupe de gestion d'objets (*Object Management Group*, OMG), le moteur de communications Internet (*Internet Communications Engine*, ICE) [20] de l'entreprise ZeroC, les langages de programmation concurrente, tels que Erlang, et les applications utilisant des services Web [21] et le protocole simple d'accès à des objets (*Simple Object Access Protocol*, SOAP).

Applications en flots de travail Une application en flot de travail comporte plusieurs programmes intercommuniquant et s'exécutant en série, en parallèle, ou les deux [22] sur plusieurs processeurs. Selon le flot de travail qui les organise, les programmes se partagent les activités de lecture des données d'entrée, de traitement de ces données, et d'écriture des données de sortie. Dans ce modèle, un programme ne peut être exécuté que quand toutes ses données d'entrée sont disponibles, c'est-à-dire quand tous les programmes qui le précèdent dans le flot ont produit leurs données de sortie. Autrement dit, les données s'écoulent entre les programmes.

Parmi les modèles d'applications en flots de travail figure par exemple le langage d'exécution de services Web de processus d'affaires (*Web Services Business Process Execution Language*, WSBPEL) [23] proposé par l'organisation pour l'avancement de standards d'information structurée (*Organization for the Advancement of Structured Information Standards*, OASIS) dont une implantation est le moteur de chef d'orchestration (*Orchestration Director Engine*, ODE) [24] pour le serveur Web Apache.

Applications à base de composants Une application à base de composants est un ensemble de composants logiciels interconnectés [25]. Un composant logiciel peut généralement être un ensemble de composants interconnectés (un assemblage) ou bien un programme. Un composant exhibe des ports à travers lesquels il consomme ou produit des données. En plus de leur orientation, les ports sont typés. Deux composants peuvent être interconnectés s'ils offrent des ports complémentaires, c'est-à-dire d'orientations contraires et de types compatibles.

Les modèles d'applications à base de composants ont les trois caractéristiques suivantes :

- La manière de spécifier un composant. Certains modèles proposent pour ce faire un langage de définition d'interfaces (*Interface Definition Language*, IDL) indépendant des langages de programmation ;
- La manière d'implanter un composant. Certains modèles spécifient pour ce faire des règles de correspondance entre leur langage de définition d'interfaces et des langages de programmation ordinaires ;
- La manière de spécifier les communications entre les composants. Certains modèles spécifient pour ce faire un langage de description d'architecture (*Architecture Description Language*, ADL) dans lequel des assemblages de composants peuvent être décrits, proposent dans leur implantation une interface de connexion de composants, ou les deux.

Parmi les modèles d'applications à base de composants figurent par exemple l'architecture commune de composants (*Common Component Architecture*, CCA) [26], le modèle de composants CORBA (*CORBA Component Model*, CCM) [27], Fractal [28], et l'architecture de composants de services (*Service Component Architecture*, SCA) [29].

Les modèles de composants permettent de programmer aussi bien des applications parallèles, paramétriques, réparties, q'en flots de travail. Par exemple, GridCCM [30] est une extension de CCM supportant des composants parallèles. Un composant parallèle GridCCM est un ensemble de composants CCM ordinaires mais strictement identiques. Héritant des propriétés de CORBA sur lequel CCM est fondé, GridCCM ajoute le parallélisme à l'hétérogénéité et la dynamicité des applications réparties.

Données, traitements, et flux

Données Les données sont des éléments d'information structurés et mémorisés (par les mémoires). Elles sont les éléments d'information qui ont été, sont, ou seront traitées par les processeurs et accédés grâce aux connecteurs. En effet, le sens commun des données exclut les programmes qui, depuis l'architecture de Von Neumann, sont pourtant également des données. Les principaux types de données sont les empreintes en mémoire vive, les fichiers en mémoire de masse, et les bases de données combinant mémoires vive et de masse.

Empreintes La gestion de l'empreinte en mémoire vive d'un programme varie selon les langages de programmation. Les langages de programmation de bas niveau, tels que Fortran et C, requièrent des développeurs qu'ils programment l'allocation et la libération de mémoire vive pour leurs programmes. Cependant, les langages de programmation de plus haut niveau, tels que Java, OCaml, et Erlang, gèrent automatiquement la mémoire vive.

Fichiers La structure des fichiers est appelée format. Parmi les formats figurent, par exemple, le format exécutable et capable de liaison (*Executable and Linkable Format*, ELF), le format de valeurs séparées par des virgules (*Coma Separated Values*, CSV), de données hiérarchiques (*Hierarchical Data Format*, HDF), le format de données commun en réseau (*Network Common Data Format*, NetCDF), le format de maillages Silo, et plus généralement les langages de balisage, tels que le langage extensible de balisage (*eXtensible Markup Language*, XML) ou encore un autre langage de balisage (*Yet Another Markup Language*, YAML).

Bases de données La structure des bases de données est appelée schéma. Le schéma le plus répandu est le schéma relationnel, introduit par Codd, fondé sur la logique des prédicats et la théorie des ensembles. Ces bases de données sont interrogées à l'aide du langage de requêtes standard (*Standard Query Language*, SQL). D'autres schémas existent, tels que le schéma objet spécifié par le groupe de gestion de bases de données objet (*Object Database Management Group*, ODBG). Parmi les implantations de systèmes de gestion de bases de données figurent par exemple MySQL (relationnelle), PostgreSQL et Mnesia [31] (objet et relationnelle).

Traitements Un traitement est une séquence d'instructions d'un programme exécutée sur un processeur. Parmi les instructions qu'il donne au processeur sur lequel il est exécuté, un traitement peut commander des connecteurs afin d'accéder aux données mémorisées. Un programme peut comporter plusieurs traitements s'exécutant simultanément, une application peut comporter plusieurs programmes s'exécutant simultanément, et une application peut exécuter simultanément plusieurs fois un même programme. L'organisation des traitements d'une application détermine son modèle : application parallèle, paramétrique, répartie, en flot de travail, ou à base de composants (voir les définitions ci-dessus). Ces modèles « purs » peuvent également être mélangés. GridCCM, par exemple, offre des composants parallèles.

Flux Un flux est un envoi, une réception, ou les deux, de données par un traitement. Les flux sont supportés par les connecteurs. Ils permettent aux traitements de mémoriser et retrouver des données d'une part, et de communiquer entre eux ou avec le monde extérieur d'autre part. L'organisation des flux d'une application permettant à ses traitements de communiquer détermine son modèle de communication : communication en mémoire partagée, en transmission de messages, voire un modèle mélangeant ces deux approches.

Partage de Mémoire Des traitements peuvent communiquer en partageant une même zone de mémoire. Parmi les technologies permettant de faire communiquer des traitements par mémoire partagée figurent les variables partagées entre des fils d'exécution POSIX, les communications inter-processus (*InterProcess Communications*, IPCs), et les mémoires partagées réparties (*Distributed Shared Memories*, DSMs) [32]. Contrairement aux variables partagées entre fils d'exécution POSIX et aux communications inter-processus, les mémoires partagées réparties impliquent plusieurs mémoires vives. Pour ce faire, leurs implantations utilisent généralement l'autre modèle de communication : le passage de messages ou un réseau logique. La mémoire juxtaposée (*Juxtaposed Memory*, JuxMem) [33] est un exemple de mémoire partagée globale utilisant un réseau logique.

Lorsque les traitements sont exécutés en parallèle, partager de la mémoire pose cependant le problème de la cohérence des flux. En effet, si deux traitements écrivent en parallèle dans la même zone de mémoire, alors les données de cette zone risquent d'être incohérentes. De plus, si les données sont d'abord mémorisées dans des mémoires tampons (mémoires caches) avant d'être effectivement mémorisées dans la mémoire partagée, la cohérence entre les mémoires tampons et la mémoire partagée peut être perdue. Des protocoles d'accès sont donc essentiels pour la communication par mémoire partagée. L'utilisation de verrous de protection et l'invalidation des données en mémoire cache sur écriture en mémoire vive sont des exemples de protocoles garantissant la cohérence des données.

Passage de messages Des traitements peuvent également communiquer en se transmettant des messages. Les messages peuvent véhiculer des signaux, des paquets de données, ou des invocations de fonctions. Parmi les technologies permettant de faire communiquer des traitements par transmission de messages figurent les signaux des communications inter-processus (IPCs), les services de publication-souscription, tels que le bus de d'ordinateur de bureau (*Desktop Bus*, D-Bus) [34], l'interface de passage de message (*Message Passing Interface*, MPI) [15], la bibliothèque de communication à haute performance Madeleine [35], les messages actifs de Berkeley (*Active Messages*, AMs) [36], le protocole de transferts hypertexte (*HyperText Transfer Protocol*, HTTP), le protocole d'accès simple à des objets (*Simple Object Access Protocol*, SOAP), les appels de procédures à distance, et les implantations du modèle d'acteurs (*Actor Model*) [37].

De même que pour les communications par mémoire partagée, afin d'être fiabilisées, les communications par transmission de messages sont régies par des protocoles. Ces protocoles suivent généralement le modèle de référence ouvert d'interconnexion de systèmes (*Open Systems Interconnection Reference Model*, OSI Reference Model) qui définit les sept couches de communication suivantes (des connecteurs constituant la couche physique aux applications) : couche physique, de liaison de données, réseau, de transport, de session, de présentation, et d'application.

Réseau logique Certains protocoles de communication par transmission de messages ont rajouté une couche entre celle de présentation et celle d'application, appelée surcouche (*overlay*). Les communications dans cette surcouche peuvent être structurées, les points communiquant peuvent être indexés, ou les deux. Parmi les technologies utilisant une surcouche figurent les technologies de réseaux privés virtuels (*Virtual Private Networks*, VPNs), tels que OpenVPN, et les applications pair à pair dont Chord [38] et les tableaux d'indexation répartie (*Distributed Hash Tables*, DHTs) sont des modèles et JXTA et Gnunet des implantations. Les traitements utilisant ces protocoles afin de communiquer peuvent ainsi le faire indépendamment des connecteurs.

Discussion

La performance de l'informatique dépend non seulement de celle de son infrastructure, mais également de l'adéquation des technologies d'applications qui sont utilisées. En effet, le nombre de technologies d'applications est grand mais toutes ne permettent pas d'obtenir des performances équivalentes. L'adéquation des technologies d'applications utilisées est double : celle par rapport au problème que l'application traite, d'une part, et celle par rapport aux technologies de ressources, d'autre part.

L'adéquation des technologies d'applications au problème traité est de la responsabilité du développeur. Pour implanter un grand nombre de traitements indépendants, celui-ci choisira par exemple une technologie d'application paramétrique. Cette adéquation peut également être renforcée par l'optimisation des algorithmes utilisés, c'est-à-dire par la maîtrise de leurs complexités

en occupation du processeur (temps) et de la mémoire vive (espace). Elle peut également évoluer avec le problème traité, que celui-ci grossisse ou se complexifie. L'augmentation du nombre de points d'un maillage et le couplage de l'application avec d'autres en sont respectivement des exemples.

L'adéquation des technologies d'applications à celles des ressources peut par exemple être remise en cause quand une application doit être exécutée sur une infrastructure informatique aux caractéristiques sensiblement différentes de celles de l'infrastructure pour laquelle elle avait été optimisée. Un tel changement peut alors nécessiter de « porter » cette application, c'est-à-dire de la reprogrammer partiellement voire totalement avec des technologies d'applications plus adaptées.

De même que les ressources sur lesquelles ils sont exécutés, les programmes peuvent être multipliés, spécialisés, et leur nombre peut varier pendant la durée de l'exécution de l'application à laquelle ils appartiennent. Afin d'améliorer leur performance, les applications peuvent donc être parallèles, hétérogènes, et dynamiques.

2.3.3 Systèmes d'exploitation

Les systèmes d'exploitation sont les éléments logiciels permettant aux applications d'être exécutées sur les ressources, autrement dit d'exploiter ces ressources pour ces applications. Après en avoir présenté les principes, cette sous-section présente des exemples de systèmes d'exploitation d'ordinateurs, de grappes, et de grilles d'ordinateurs. Puis elle présente les trois types de services d'un système d'exploitation que sont la mémorisation, l'exécution, et l'adressage.

Principes

Les systèmes d'exploitation et les applications sont des éléments logiciels séparés. Cette séparation reproduit celle des responsabilités des administrateurs et des utilisateurs. Les administrateurs sont responsables des ressources et les mettent à disposition d'utilisateurs en exécutant sur elles des systèmes d'exploitation. Les utilisateurs doivent utiliser ces systèmes afin d'exécuter leurs applications sur les ressources.

Les systèmes d'exploitation manipulent les trois concepts suivants : des ressources, des applications, et des utilisateurs. Afin d'exploiter les ressources, ils peuvent intégrer des éléments logiciels spécialisés, appelés pilotes, souvent fournis avec les ressources par leurs constructeurs. Ils offrent des interfaces de programmation d'application (*Application Programming Interface*, API) utilisant ces ressources, interfaces qui peuvent être spécialisées ou standardisées. L'interface de système d'exploitation portable pour Unix (*Portable Operating System for unIX*, POSIX) et l'API simple pour des applications de grille (*Simple API for Grid Applications*, SAGA) [39] sont des exemples d'interfaces standardisées.

Les interfaces de programmation d'applications des systèmes d'exploitation sont également offertes par des éléments logiciels interprêtant des commandes, ces éléments étant appelés `shells`. Un utilisateur peut soumettre ces commandes de manière interactive ou bien il peut les inscrire dans un fichier appelé script pour une soumission différée. sh, ksh, tcsh, GNU bash, et zsh sont des exemples de `shells`.

Une application en cours d'exécution dans un système d'exploitation est un ensemble de processus. Un processus comporte un ou plusieurs traitements, également appelés fils d'exécution, une empreinte en mémoire vive, et les descripteurs des fichiers auxquels il accède. À chaque processus correspondent un programme, une commande comportant le nom du programme et d'éventuels paramètres, un identifiant de processus et d'utilisateur, voire un identifiant de groupe de processus et de session d'utilisateur. Différentes techniques permettent aux processus d'intercommuniquer, qu'ils fassent partie d'une même application ou non.

Un système organise le partage, entre les processus, des ressources qu'il exploite. Pour organiser le partage des processeurs, sans lesquels l'exécution des processus ne peut continuer, il comporte un élément logiciel appelé ordonnanceur qui prescrit les allocations des processeurs aux processus. Il restreint également l'accès aux programmes des administrateurs par un mécanisme de gestion de permissions. Enfin, il peut comptabiliser et faire respecter des quotas d'accès aux ressources et particulièrement aux mémoires de masse.

Exemples de systèmes d'exploitation

Système d'exploitation des super-ordinateurs Blue Gene L'exploitation des super-ordinateurs Blue Gene est effectuée à deux niveaux. Le premier niveau comporte les nœuds, spécialisés soit pour le calcul, soit pour les entrées-sorties. Les premiers sont exploités par des systèmes à micro-noyau ne supportant qu'un seul traitement à la fois, et les seconds sont exploités par des systèmes GNU/Linux [40]. Au second niveau d'exploitation se trouve une station de travail, également exploitée par GNU/Linux, sur laquelle est exécuté l'ordonnanceur (LoadLeveler) d'IBM. Toutes les applications doivent passer par ce gestionnaire. Les services de mémorisation sur la baie externe, pilotés par le système de fichiers général et parallèle d'IBM (*General Parallel File System*, GPFS) [41], sont rattachés aux nœuds d'entrée-sortie. Enfin les services d'adressage au travers des connecteurs à haute performance sont matériels et propriétaires.

Système d'exploitation des grappes Beowulf De même que celle de Blue Gene, l'exploitation des grappes d'ordinateurs de type Beowulf est effectuée à deux niveaux. Les nœuds de calcul et le nœud frontal sont exploités par des systèmes GNU/Linux et un gestionnaire de lots, tel que le gestionnaire de ressources et de files d'attente (*Terascale Open-Source Resource and QUEue Manager*, TORQUE). Les nœuds de calcul disposent de services de mémorisation locaux pour des fichiers temporaires et se partagent un service de mémorisation, tel qu'un système de fichiers mis en réseau (*Networked File System*, NFS) ou Lustre [42].

Systèmes à image unique Les systèmes (d'exploitation) à image unique (*Single System Images*, SSIs) offrent une interface de programmation unique pour des ressources faisant partie d'infrastructures qui auraient pu être indépendantes. De tels systèmes offrent donc des services de mémorisation, d'exécution, et d'adressage globaux pour des architectures de type grappes ou grilles d'ordinateurs. Parmi ces systèmes figurent par exemple l'infrastructure de migration de processus pour Linux (*Linux Process Migration Infrastructure*, LinuxPMI) et Kerrighed [43] pour la grappe, Vigne [44] et XtreemOS [45] pour la grille.

Boîte à outils Globus Globus est une boîte à outils qui fournit les services nécessaires pour l'exploitation d'une grille d'ordinateurs [46], infrastructure informatique définie par Ian Foster dans son article « Qu'est-ce que la grille ? Une liste de contrôle en trois points » (*What is the grid ? A Three Point Checklist*) [47]. Le service d'exécution est décomposé en plusieurs services : le langage de spécification de ressources (*Resource Specification Language*, RSL), le gestionnaire d'allocation de ressources Globus (*Globus Resource Allocation Manager*, GRAM), et le portier (*Gatekeeper*) qui permet l'accès aux ressources. Le système de suivi et de découverte (*Monitoring and Discovery System*, MDS) est le service d'information statique et dynamique des ressources exploitées par Globus. Le service de mémorisation est décomposé en plusieurs services fondés sur le protocole de transfert de fichiers de grille (*Grid File Transfer Protocol*, GridFTP) : l'accès global à la mémorisation secondaire (*Global Access to Secondary Storage*, GASS), le transfert fiable de fichiers (*Reliable File Transfer*, RFT), et le service de localisation de répliques (*Replica Location Service*, RLS).

Mémorisation, exécution, et adressage

Mémorisation Lors de la préparation de l'exécution d'applications voire tout au long de celle-ci, les processeurs copient en mémoire vive les programmes et les données à traiter à partir des mémoires de masse dans lesquelles ils sont mémorisés. Réciproquement, ces processeurs peuvent également copier les données de la mémoire vive vers les mémoires de masse, particulièrement à la fin de l'exécution d'applications. L'accès à ces données et leur mémorisation sont organisés par un élément logiciel du système d'exploitation, le service de mémorisation. Ce service comporte à la fois les outils permettant de copier, déplacer, supprimer, etc. les données, les mécanismes de gestion des permissions et des quotas, et les systèmes (de gestion) de fichiers.

En mémoire de masse, les données sont inscrites dans des fichiers organisés en arborescence. Un répertoire de fichiers peut effectivement contenir des fichiers, d'autres répertoires de fichiers, ou les deux. Chaque élément de l'arborescence a un nom, un chemin absolu dans l'arborescence, et des attributs, tels que l'identifiant de l'utilisateur propriétaire des données, les permissions accordées pour l'accès à ces données, et les dates de création et des derniers accès à ces données. Cette arborescence constitue l'espace de désignation de fichiers du système d'exploitation. Dans la famille de systèmes Unix, cette arborescence a d'ailleurs été standardisée (*Filesystem Hierarchy Standard*, FHS) [48].

En tant que sous-systèmes d'exploitation, les services de mémorisation ont une architecture. Un service de stockage peut ne pas modifier directement des fichiers mais plutôt mémoriser les modifications demandées dans un fichier spécial, appelé journal, avant de les effectuer. On parle alors de service de mémorisation journalisée. Un service de stockage peut gérer plusieurs mémoires de masse et peut également être utilisé par plusieurs systèmes d'exploitation. Dans le premier cas, il s'agit d'augmenter la taille disponible en ajoutant les tailles de chacune, d'augmenter le débit en ajoutant les débits de chacune, ou les deux. On parle alors respectivement de service de mémorisation répartie, parallèle, ou les deux. Dans le second cas, il s'agit de disposer d'un espace de désignation unique et on parle alors de service de mémorisation partagée.

Parmi les services de mémorisation figurent par exemple le système de fichiers virtuel (*Virtual File System*, VFS) et le système de fichiers journalisé EXT4 de GNU/Linux, le système de fichiers partagé en réseau (*Networked File System*, NFS), le système de fichiers virtuel parallèle (*Parallel Virtual File System*, PVFS) [49], et le système de fichiers réparti et parallèle Lustre [42].

Exécution Afin d'exécuter les applications qu'on leur soumet, les systèmes d'exploitation comportent un service d'exécution qui organise le partage des ressources à différents niveaux et à la fois dans le temps et dans l'espace. Pour le partage dans le temps, à bas niveau les traitements utilisent les processeurs par intermittence. À haut niveau, avant d'être exécutées les applications peuvent patienter dans des files d'attente. Pour le partage dans l'espace, à bas niveau des traitements sont exécutés simultanément sur différents processeurs et se partagent éventuellement l'accès à des mémoires vives. À haut niveau, les applications sont exécutées sur des partitions de l'ensemble des ressources.

Les services d'exécution comportent les cinq parties suivantes :

- Une interface de soumission d'applications ;
- Un mécanisme de gestion de files d'attente ;
- Un ordonnanceur ;
- Un mécanisme d'exécution d'applications ;
- Un mécanisme de suivi de processus sur des processeurs.

Les administrateurs décident de la politique d'exécution mise en œuvre par l'ordonnanceur du service d'exécution, entre une exécution immédiate sur des ressources éventuellement déjà utilisées ou une exécution différée sur des ressources réservées. Le service d'exécution des systèmes d'exploitation des ordinateurs, tel que celui de GNU/Linux, le service d'ordonnancement des

systèmes à image unique, et les outils de soumission de commande à distance, tels que le `shell` sécurisé (*Secure SHell*, SSH), mettent en œuvre une politique d'exécution immédiate. En revanche des services d'exécution supplémentaires, tels que les gestionnaires de lots, mettent en œuvre une politique d'exécution différée. Ces gestionnaires sont exécutés avec le système d'exploitation des ordinateurs qu'ils exploitent, ils ajoutent donc un niveau dans le service d'exécution global.

Les gestionnaires de lots, produits afin d'exploiter les grappes d'ordinateurs, comportent généralement deux parties. La première, appelée serveur, implante l'interface de soumission, le mécanisme de gestion de files d'attente, et l'ordonnanceur. La seconde, appelée client, implante les mécanismes d'exécution et de suivi des processus. Le serveur est généralement exécuté sur le nœud frontal et les clients sur les nœuds de calcul, ces derniers communiquant avec le premier par le réseau. Parmi les gestionnaires de lots figurent par exemple le niveleur de charge (*LoadLeveler*) d'IBM, TORQUE, et le moteur de grille N1 (*N1 Grid Engine*) de Sun Microsystems.

Adressage Lorsque des traitements sont exécutés sur des processeurs, ils accèdent à l'empreinte en mémoire vive des processus auxquels ils appartiennent, lisent et écrivent des fichiers mémorisés en mémoire de masse, ou communiquent avec d'autres traitements en émettant et recevant des messages sur le réseau, le tout en respectant les protocoles des connecteurs impliqués dans ces flux. Pour ce faire, les systèmes d'exploitation comportent des services d'adressage.

L'adressage de zones de mémoire vive utilise généralement le mécanisme de mémoire virtuelle dans lequel les adresses logiques, manipulées par les traitements, sont traduites en adresses physiques, manipulées par les connecteurs. Ce mécanisme permet au système d'exploitation d'exploiter la mémoire vive comme il le souhaite et, particulièrement, de déplacer le contenu de zones de mémoire vive sans en avertir les traitements. Ce mécanisme permet également d'utiliser des mémoires de masse quand la mémoire vive est pleine (*swapping*).

L'adressage de zones de mémoire de masse utilise la structure virtuelle de celle-ci. Dans GNU/Linux, cette structure est appelée le système de fichiers virtuel (*Virtual File System*, VFS). Il comporte notamment des « superblocs », des « i-nœuds », et des « d-entrées ». Les superblocs représentent les systèmes de fichiers chargés (*mounted filesystems*). Les i-nœuds correspondent aux données, telles que les fichiers. Enfin, les entrées de répertoire, ou dentrées, représentent les nœuds de l'arborescence. Les fonctions d'accès aux fichiers de l'interface de programmation de GNU/Linux manipulent ces éléments et, par eux, les implantations de différents systèmes de fichiers.

L'adressage d'éléments communiquant est également assuré par le système d'exploitation. Dans GNU/Linux, les communications inter-processus sont implantées dans le noyau et les protocoles de communication avec d'autres ordinateurs en réseau dans des extensions du noyau appelées modules. La suite de protocoles d'Internet par exemple, tels que le protocole de contrôle de transmission (*Transmission Control Protocol*, TCP) et le protocole Internet (*Internet Protocol*, IP), et le protocole de communication Infiniband, appelé OpenFabrics, sont implantés ainsi. Les systèmes d'exploitation de la famille de la distribution logicielle de Berkeley (*Berkeley Software Distribution*, BSD), tels que FreeBSD, sont particulièrement reconnus pour la fiabilité des implantations de ces protocoles.

Discussion

En permettant aux applications d'utiliser les ressources, les systèmes d'exploitation ont un rôle déterminant. Les services de mémorisation, d'exécution, et d'adressage qu'ils comportent sont en effet appelés très fréquemment. Deux logiques peuvent alors s'affronter : l'une consistant à optimiser le système d'exploitation pour certaines technologies de ressources et d'applications, l'autre à laisser au système d'exploitation sa généricité initiale. De nombreuses infrastructures

choisissent cette seconde logique en ayant recours au système d'exploitation d'ordinateurs ordinaires. En revanche, d'autres infrastructures utilisent des micro-noyaux plus difficiles à utiliser mais optimisés.

On constate également que malgré des évolutions permanentes des technologies de ressources et d'applications, les systèmes d'exploitation, ou du moins leurs interfaces, sont relativement stables. Par exemple, bien que son code source évolue quotidiennement, le système d'exploitation GNU/Linux reste clairement membre de la famille Unix et, comme tel, implante l'interface de programmation POSIX qui évolue peu. En les implantant afin d'exploiter des grappes ou des grilles d'ordinateurs, les systèmes à image unique renforcent ce phénomène de stabilité des interfaces.

La taille des ressources et des applications leur confère des propriétés de parallélisme, d'hétérogénéité, et de dynamicité qu'elles n'avaient pas quand ont été définies les interfaces des systèmes d'exploitation qui les exploitent et les exécutent. Il en résulte la multiplication des technologies d'applications sous la forme d'intergiciels (*middleware*) ou de cadriciels (*frameworks*).

2.4 Discussion

Par rapport à celle de Flynn, la classification présentée ci-dessus propose quatre évolutions. La première est de détailler la transmission entre l'opération et l'information. La deuxième est de distinguer l'homogénéité et l'hétérogénéité des ensembles d'éléments. La troisième est de considérer les éventuelles variations temporelles des cardinalités de ces ensembles. La quatrième est de distinguer les ressources, les systèmes d'exploitation, et les applications. Ces évolutions sont importantes afin de permettre au chapitre suivant de poser la problématique du déploiement.

Derrière la multitude des technologies de ressources, d'applications, et de systèmes d'exploitation introduites dans la section précédente se trame la structure générale de l'informatique : une hiérarchie d'information, d'opération, et de transmission. Les architectures de ressources matérialisent cette hiérarchie que l'on retrouve également dans les architectures logicielles, particulièrement avec les modèles d'applications à base de composants.

Bien que les définitions de ressources, de systèmes d'exploitation, et d'applications découlent de celle d'un ordinateur, la séparation entre ces trois classes de technologies ne correspond toutefois pas nécessairement à la séparation entre éléments matériels, logiciels à bas niveau, et logiciels à haut niveau. Des applications peuvent effectivement être perçues comme des ressources par certains, des ressources comme des applications par d'autres, etc. Cette classification situe cependant les différentes couches de technologies les unes par rapport aux autres et fixe un repère en croisant la séparation des rôles entre administrateurs et utilisateurs d'une part, et celle des éléments matériels et logiciels d'autre part.

2.5 Conclusion

La classification proposée dans ce chapitre a présenté les principales architectures de ressources, les principaux modèles d'applications, et des exemples de systèmes d'exploitation. Elle a cité plus d'une centaine de technologies de l'état de l'art dans le domaine de l'informatique à haute performance. Cette classification a également montré que les facteurs de performance de l'informatique conférait parallélisme, hétérogénéité, et dynamicité aux ressources comme aux applications, et que les interfaces des systèmes d'exploitation étaient stables par rapport aux technologies de ressources et d'applications.

Bien que les infrastructures informatiques les plus performantes du monde appartiennent toujours aux grandes organisations également détentrices d'expertise, telles que des institutions

de recherche académique ou des entreprises industrielles, les technologies de l'informatique à haute performance se diffusent progressivement dans la plupart des activités humaines. Les ordinateurs actuellement destinés au grand public des pays dominants ont par exemple des mémoires, des processeurs généraux multi-cœur, des processeurs graphiques et pour le réseau, et des connecteurs les interconnectant tous de plus en plus performants. Cette diffusion des technologies ne doit cependant pas masquer les difficultés d'utilisation qu'elles soulèvent, en particulier du fait du développement de leurs propriétés de parallélisme, d'hétérogénéité, et de dynamicité.

Chapitre 3

Déploiement dans le contexte de l'informatique à haute performance

3.1 Introduction

Disposer de ressources, d'applications, et de systèmes d'exploitation à haute performance ne suffit pas. Encore faut-il utiliser ces systèmes d'exploitation afin d'allouer une partie de ces ressources à ces applications, installer ces applications, et les exécuter. La classification des technologies d'informatique à haute performance, proposée dans le chapitre précédent, sépare ces trois classes de technologies et en illustre la grande diversité. En revanche elle ne traite pas de ce qui incombe à quiconque voudrait les utiliser.

Ce chapitre complète la classification du précédent en ajoutant une quatrième classe, la classe des technologies de déploiement. Il commence par proposer une définition du processus de déploiement, puis il illustre cette définition par des exemples. Il poursuit en présentant les approches actuelles afin d'effectuer les étapes de déploiement l'une après l'autre d'abord, puis l'ensemble du processus. Enfin il discute les limites de ces approches.

3.2 Cycle de vie des applications sur les ressources

Le cycle de vie d'une application, illustré par la figure 3.1 page 44, comporte deux étapes : le développement et le déploiement.

3.2.1 Développement

Le développement d'une application est un processus de production qui incombe à ses développeurs. Regroupés sous ce terme, les développeurs d'une application peuvent comporter des chercheurs, des ingénieurs, des analystes programmeurs, des utilisateurs testeurs, des éditeurs, etc. Le développement se déroule sur les ressources de ces développeurs et comporte les cinq étapes suivantes :

Spécification Spécifier une application est la décrire selon des aspects fonctionnels et non fonctionnels, c'est-à-dire la décrire d'un point de vue externe.

Conception Concevoir une application est la décrire selon des aspects technologiques, c'est-à-dire la décrire d'un point de vue interne.

Réalisation Réaliser une application est mettre en œuvre sa conception en respectant sa spécification, c'est-à-dire la programmer.

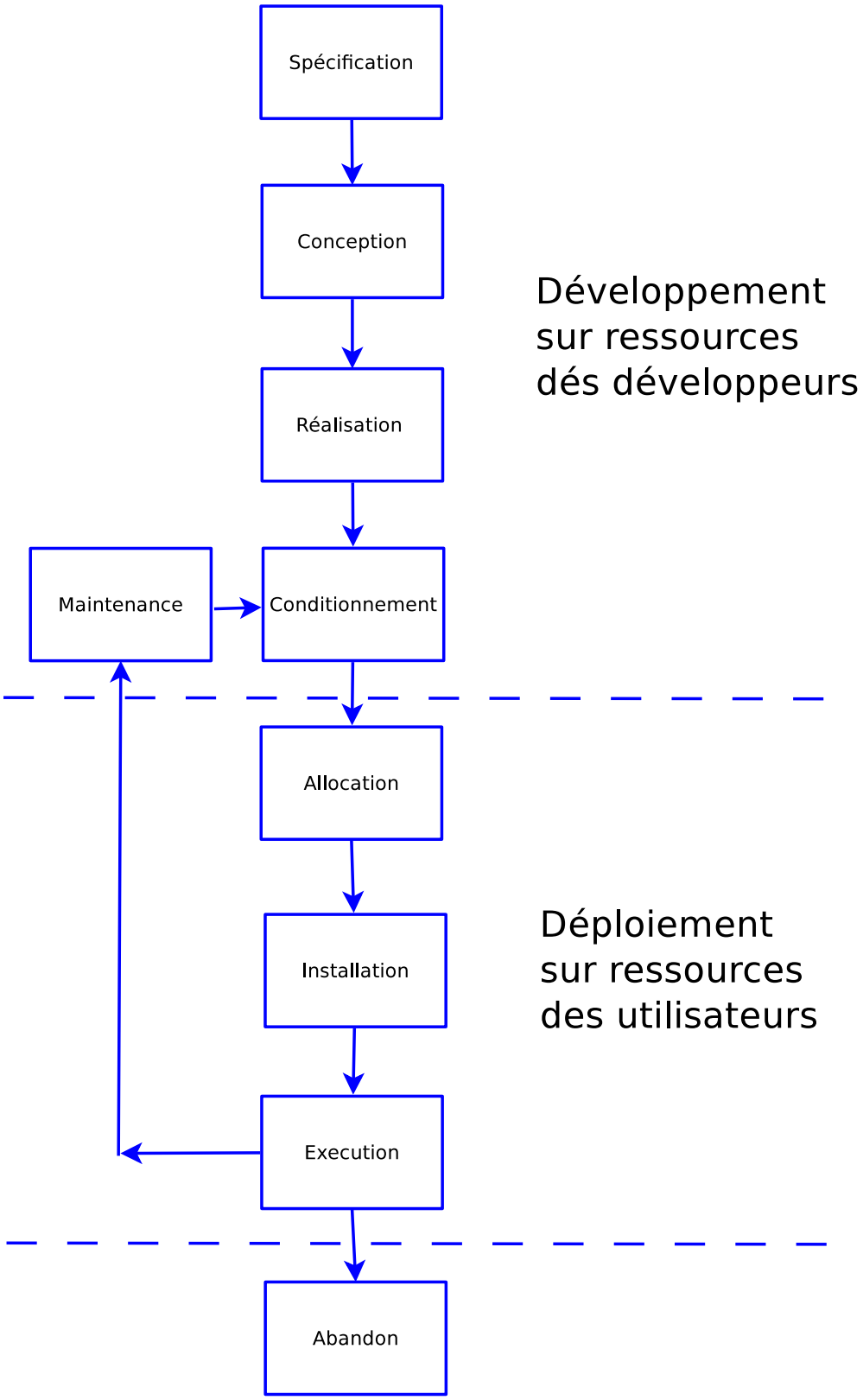


FIGURE 3.1 – Cycle de vie d'une application sur des ressources

Conditionnement Conditionner une application est la préparer afin de l'installer sur les ressources des utilisateurs, c'est-à-dire utiliser un système d'archivage, de paquetage, ou bien de composant logiciel.

Maintenance ou bien abandon Maintenir une application est l'améliorer, c'est-à-dire corriger ses erreurs, lui ajouter de nouvelles fonctionnalités, mieux la documenter, etc. L'abandonner est le contraire de la maintenir.

Les étapes de conception, de réalisation, de conditionnement, et de maintenance fixent les technologies de l'application. En effet, en concevant une application, les développeurs décident de ses modèles de données, de traitements, et de communication. En la réalisant puis en la maintenant, ils choisissent ses dépendances vers des codes externes ainsi que les systèmes d'exploitation et les architectures des ressources sur lesquels elle sera exécutée. En la conditionnant enfin, ils choisissent la manière avec laquelle elle sera installée. Ces étapes déterminent donc le processus de déploiement.

3.2.2 Déploiement

Le déploiement d'une application est un processus qui la mène d'un état de conditionnement à un état de fonctionnement, processus qui incombe à ses utilisateurs. Le processus inverse est appelé repliement. Regroupés sous ce terme, les utilisateurs d'une application peuvent comporter des développeurs, des administrateurs de ressources, des installateurs, des utilisateurs finaux, etc. Le déploiement se déroule sur les ressources de ces utilisateurs et comporte les trois étapes suivantes :

Allocation Allouer des ressources à une application est découvrir, sélectionner, réserver, ou négocier des ressources dont elle aura besoin afin d'être installée et exécutée, c'est-à-dire la placer sur des ressources afin de continuer le processus de son déploiement.

Installation Installer une application est l'acheminer, la déconditionner, la compiler, la paramétrer, ou la configurer sur les ressources qui lui ont été allouées afin d'être exécutée, c'est-à-dire la préparer afin de continuer le processus de son déploiement.

Exécution Exécuter une application est l'activer, la lancer, ou encore la démarrer afin qu'elle utilise les ressources sur lesquelles elle a été installée et qu'elle produise les résultats pour lesquels elle a été développée et ainsi terminer le processus de son déploiement.

Une intuition afin de bien comprendre la séparation entre ces trois étapes serait la suivante. Une application serait un ensemble de fichiers et de processus. Lui allouer des ressources consisterait à lui choisir des ordinateurs ayant des mémoires et des processeurs adéquats, l'installer consisterait à produire ces fichiers dans les mémoires de ces ordinateurs, et l'exécuter consisterait à produire ces processus dans les processeurs de ces ordinateurs. La figure 3.2 page 46 illustre cette intuition.

On trouve parfois le terme « allocation » utilisé dans un sens inverse que celui utilisé dans ce document. Dans ce sens inverse, l'allocation porterait sur les applications pour les ressources et dénote ainsi la même notion d'association mais du point de vue des administrateurs des ressources recevant des applications à déployer. Ce document portant sur le déploiement du point de vue des utilisateurs, on a retenu le sens donné ci-dessus, c'est-à-dire l'allocation de ressources pour les applications.

Le processus de déploiement est réglé par deux principes : la compatibilité des ressources et des applications, d'une part, et la faisabilité de l'allocation de ressources, de l'installation des applications, et de leur exécution grâce aux systèmes d'exploitation des ressources, d'autre part. Respecter ces principes, dans le cas d'une station de travail et d'une application séquentielle, est simple. Pour ce faire, il suffit de vérifier que les technologies de la station et de l'application

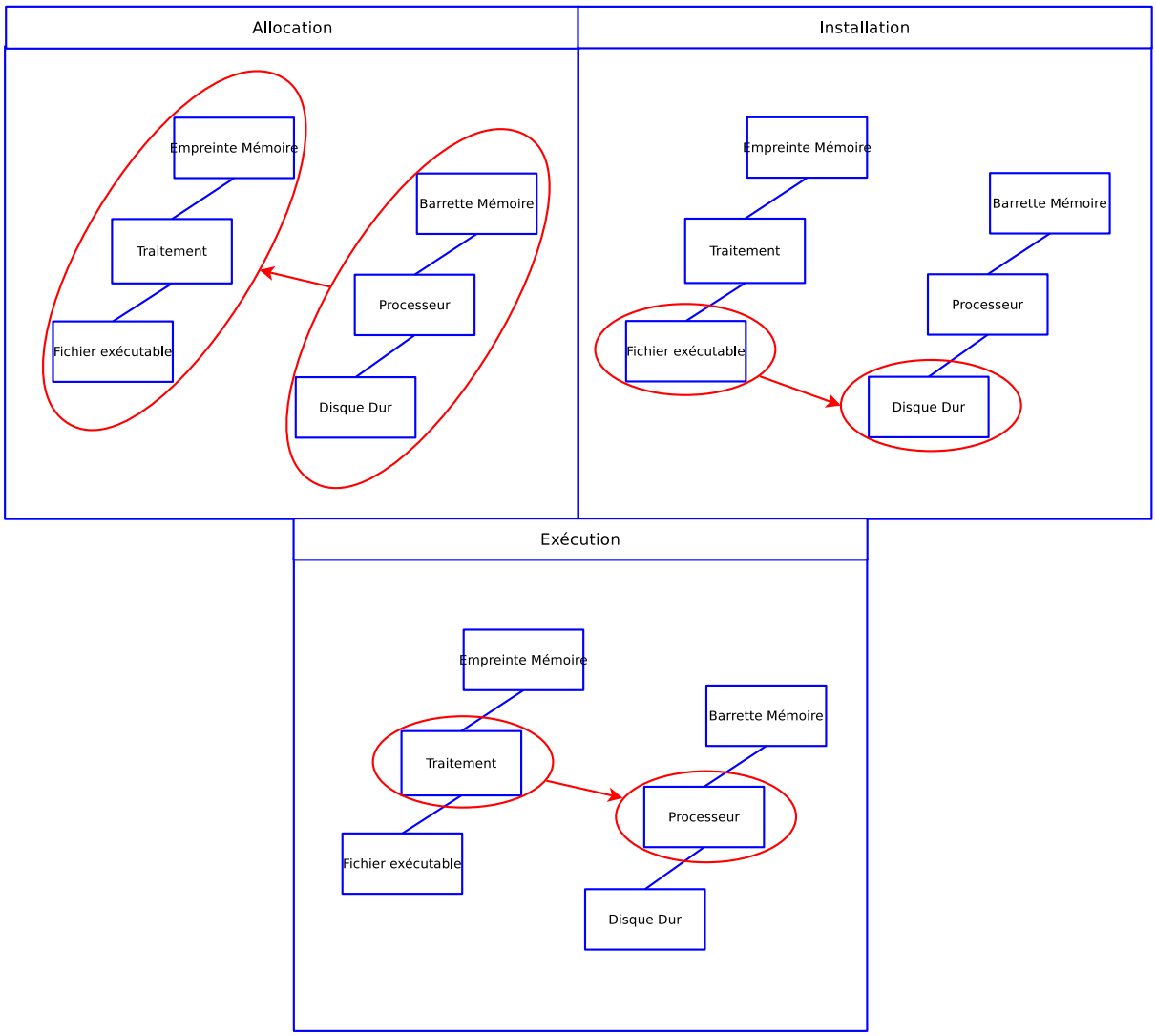


FIGURE 3.2 – Intuition du processus de déploiement

	Compatibilité	Faisabilité
Parallélisme	Correspondance topologique simple	Support technologique réparti
Hétérogénéité	Correspondance topologique complexe	Support technologique multiple
Dynamicité	Correspondance topologique variable	Support technologique adaptatif

TABLE 3.1 – Problèmes du déploiement en informatique à haute performance

soient compatibles, et que le système d'exploitation de la station permette l'allocation de la station, l'installation de l'application, et son exécution. La plupart des utilisateurs finaux savent résoudre ce problème, ne serait-ce parce qu'ils le résolvent souvent pour leur usage en informatique personnelle. Pour le déploiement dans le contexte de l'informatique à haute performance en revanche, respecter ces principes est complexe.

3.2.3 Contexte de l'informatique à haute performance

Dans le contexte de l'informatique à haute performance, les propriétés de parallélisme, d'hétérogénéité, et de dynamicité des ressources et des applications complexifient le respect des principes de compatibilité et de faisabilité du processus de déploiement. Cette complexité est essentiellement due à la multiplicité des éléments des ressources et des applications. La table 3.1 page 47 nomme chacun des problèmes présentés ci-dessous.

Respect du principe de compatibilité

Afin de respecter le principe de compatibilité, les trois propriétés des ressources et des applications posent respectivement les trois problèmes suivants :

Correspondance topologique simple Des parties du réseau des ressources doivent correspondre aux réseaux des applications. Les éléments intercommuniquant des ressources et ceux des applications étant respectivement homogènes, la compatibilité d'un élément des ressources et d'un élément des applications est donc simple à vérifier. Par exemple, si deux ordinateurs sont similaires et qu'ils sont interconnectés, d'une part, et si deux processus sont similaires et qu'ils intercommuniquent, d'autre part, alors le réseau des ordinateurs et celui des processus sont compatibles si un des ordinateurs et un des processus le sont.

Correspondance topologique complexe Le devoir de correspondance est le même que pour les topologies simples, en revanche les éléments intercommuniquant des ressources et ceux des applications étant respectivement hétérogènes, la compatibilité d'un élément des ressources et d'un élément des applications est donc plus difficile à vérifier. Par exemple, si deux ordinateurs sont différents et qu'ils sont interconnectés, d'une part, et si deux processus sont différents et qu'ils intercommuniquent, d'autre part, alors le réseau des ordinateurs et celui des processus sont compatibles si un des ordinateurs et un des processus le sont, d'une part, et si l'autre ordinateur et l'autre processus le sont également.

Correspondance topologique variable Une partie du réseau des ressources pouvant être ajoutée ou retirée, d'une part, et des parties des réseaux des applications pouvant également être ajoutées ou retirées, d'autre part, le devoir de correspondance entre des parties du premier et des seconds est donc difficile à respecter. Par exemple, si deux ordinateurs interconnectés sont disponibles par intermittence, d'une part, et si deux processus intercommuniquent, d'autre part, alors le réseau des ordinateurs et celui des processus sont compatibles si les deux ordinateurs sont disponibles simultanément.

Respect du principe de faisabilité

De même, afin de respecter le principe de faisabilité, les trois propriétés des ressources et des applications posent respectivement les trois problèmes suivants :

Support technologique réparti Les ressources allouées à une application pouvant être exploitées par des systèmes distincts, un déploiement peut impliquer d'utiliser simultanément plusieurs de ces systèmes.

Support technologique multiple Les éléments intercommuniquant des ressources pouvant avoir des technologies différentes, de même pour ceux des applications et pour les systèmes d'exploitation, un déploiement peut impliquer de manipuler simultanément plusieurs technologies.

Support technologique adaptatif Les éléments intercommuniquant des ressources et des applications pouvant varier en nombre, un déploiement peut impliquer de changer l'allocation, l'installation, ou l'exécution.

La section suivante illustre tous ces problèmes par des exemples.

3.3 Exemples de déploiement

Dans le contexte de l'informatique à haute performance, les utilisateurs sont confrontés à des cas très variés de déploiement d'applications sur des ressources (voir la classification du chapitre précédent). Parmi cette variété, cette section en a sélectionné trois particulièrement représentatifs. Il s'agit du déploiement des trois modèles d'applications suivants respectivement sur les trois architectures de ressources suivantes :

- Les applications paramétriques, parallèles, et à base de flots de travail ;
- Les ordinateurs, les grappes d'ordinateurs, et les grilles d'ordinateurs.

Cette section présente chacun des trois cas de déploiement retenus, du plus simple au plus complexe. Pour chaque cas, elle précise quels sont les ressources et leurs systèmes d'exploitation, l'application, et les conditions initiales. Elle décrit ensuite chaque étape du processus de déploiement de l'application sur les ressources, compte-tenu des conditions initiales.

3.3.1 Application paramétrique sur ordinateur

Le premier exemple est le processus de déploiement d'une application paramétrique sur un ordinateur distant.

Cas de déploiement

Un utilisateur veut déployer son application paramétrique et il a accès à deux ordinateurs distants. Les détails de la situation dans laquelle il se trouve suivent.

Ordinateurs Les ordinateurs auxquels l'utilisateur a accès sont deux stations de travail multiprocesseur. La première a deux processeurs AMD 64 bits avec chacun quatre cœurs. Elle a également 2 Go de mémoire vive et un disque dur SCSI sur lequel se trouve le répertoire personnel de l'utilisateur ainsi qu'un répertoire d'écriture temporaire `scratch`, de 100 Go chacun. La seconde a quatre processeurs I686 32 bits ayant chacun deux cœurs. Elle a également 4 Go de mémoire vive et trois disques durs SATA en RAID5 sur lesquels se trouve le répertoire personnel de l'utilisateur ainsi qu'un répertoire d'écriture temporaire `tmp`, de 50 Go chacun. Ces deux ordinateurs sont interconnectés au sein d'un réseau TCP/IP local. Ils fonctionnent sous GNU/Linux 2.6.20 et acceptent des commandes par SSH et des transferts de fichiers par SCP.

Application paramétrique L'application comporte trois éléments : un lanceur, un code paramétrique, et un jeu de données à traiter. Le lanceur est un script `shell` qui construit et soumet localement, simultanément, toutes les commandes pour les cas d'étude impliquant chacun le code paramétrique lui-même. Ses paramètres sont le chemin d'accès au code paramétrique, le chemin d'accès au jeu de données à traiter, et des paramètres spécifiques de l'étude paramétrique. Le code paramétrique lui-même est un programme binaire GNU/Linux I686 32 bits ayant deux fils d'exécution et dont les paramètres sont le chemin d'accès au jeu de données à traiter, le chemin d'accès au jeu de données traitées, et des paramètres spécifiques à un cas de l'étude paramétrique. Enfin, le jeu de données à traiter est un fichier de 10 Go.

Conditions initiales L'ordinateur de l'utilisateur est connecté au sein du réseau local dans lequel se trouvent les ordinateurs de calcul. Les fichiers de l'application paramétrique se trouvent dans un sous-répertoire `etude_parametrique` du répertoire personnel de l'utilisateur, sur son ordinateur. A priori, l'application est compatible avec un des ordinateurs de calcul. En revanche le nombre de codes paramétriques à exécuter pour l'étude considérée dépasse les capacités des ordinateurs, c'est-à-dire que tous ces codes ne pourront pas être exécutés parallèlement.

Processus de déploiement

Afin de déployer son application paramétrique, l'utilisateur suit les trois étapes du processus de déploiement que sont l'allocation, l'installation, et l'exécution. Les détails de ces étapes suivent.

Allocation Afin d'allouer un ordinateur pour son application, l'utilisateur doit connaître les besoins de son application et les capacités des ordinateurs auxquels il a accès. Dans le meilleur des cas, il dispose de cette information grâce aux développeurs de son application et aux administrateurs des ordinateurs. Dans le pire des cas, il scrute son application et les ordinateurs eux-mêmes. De toute manière, une fois l'information obtenue, il vérifie la compatibilité de son application et d'un ordinateur : il considère un autre ordinateur tant que cette compatibilité est mauvaise. N'ayant considéré que les ordinateurs auxquelles il avait accès, une fois l'ordinateur compatible sélectionné, l'utilisateur respecte les deux principes du déploiement que sont la compatibilité et la faisabilité. En l'occurrence, cet ordinateur serait le second.

Les processeurs sont programmés dans un langage de bas niveau propre à leur architecture. Par conséquent, si le jeu d'instructions utilisé par un programme donné diffère de celui offert par un processeur donné, alors ce programme risque de ne pas fonctionner directement sur ce processeur. Le cas échéant et si cela s'avérait opportun, un programme dédié pourrait être écrit en utilisant le jeu d'instructions du processeur afin d'offrir un autre jeu d'instructions, on parle alors d'émulation voire de virtualisation. Toutefois, en augmentant le nombre d'appels aux instructions pour fonctionner, un tel programme restreint la puissance de calcul disponible. Par exemple un programme utilisant un jeu d'instructions I686 ne peut pas fonctionner directement sur un processeur ayant un jeu d'instructions AMD64. Un tel programme aurait besoin d'un émulateur de processeur I686 pour AMD64.

Installation Afin d'installer son application sur l'ordinateur alloué, l'utilisateur doit d'abord transférer le sous-répertoire de cette application depuis son répertoire personnel sur son ordinateur vers celui sur l'ordinateur alloué. Pour ce faire il doit utiliser le protocole de transfert de fichiers disponible sur cet ordinateur : SCP. L'utilisateur doit ensuite considérer les nouveaux chemins d'accès aux répertoires de son application et d'écriture temporaire, dans la perspective de l'exécution de son application. En effet les chemins d'accès à son répertoire personnel et au

répertoire d'écriture temporaire sur l'ordinateur alloué peuvent être différents de ceux sur son ordinateur.

Exécution Afin d'exécuter son application, l'utilisateur doit d'abord soumettre à distance la commande pour le lanceur de son application. Pour ce faire, il doit construire la commande comportant les chemins d'accès correspondant à l'installation sur l'ordinateur alloué puis utiliser le protocole de soumission à distance de commandes sur cet ordinateur : SSH. Ensuite l'utilisateur doit scruter la fin de l'exécution de son application et transférer le fichier de données traitées depuis l'ordinateur alloué vers son ordinateur, à nouveau grâce à SCP.

Dans ce cas, bien que cela soit particulièrement fastidieux, respecter les principes de compatibilité et de faisabilité afin de déployer est à la portée de nombreux utilisateurs finaux. En revanche, le cas suivant est plus complexe pour eux.

3.3.2 Application parallèle sur grappe d'ordinateurs

Le deuxième exemple est le processus de déploiement d'une application parallèle sur une grappe d'ordinateurs distants.

Cas de déploiement

Un utilisateur veut déployer son application parallèle et il a accès à deux grappes d'ordinateurs distants. Les détails de la situation dans laquelle il se trouve suivent.

Grappes d'ordinateurs Les ordinateurs auxquels l'utilisateur a accès sont deux grappes de stations de travail multi-processeur. La première grappe comporte neuf ordinateurs bi-processeur quadri-cœur AMD 64 bits, du type de celui décrit précédemment. Ces ordinateurs sont interconnectés au sein de deux sous-réseaux commutés Ethernet 100 Mbps et Infiniband. Un de ces ordinateurs a un disque dur supplémentaire de 500 Go sur lequel se trouvent les répertoires personnels des utilisateurs et les exporte par NFS vers tous les autres ordinateurs, qui doivent ainsi se les partager. La seconde grappe comporte 17 ordinateurs quadri-processeur bi-cœur I686 32 bits, du type de celui décrit précédemment. De même que dans la première grappe, un de ces ordinateurs a un disque dur supplémentaire de 200 Go sur lequel se trouvent les répertoires personnels des utilisateurs et les exporte par NFS vers tous les autres ordinateurs, qui doivent ainsi se les partager. Ces ordinateurs sont interconnectés au sein d'un réseau commuté Ethernet 100 Mbps. Ces deux grappes d'ordinateurs sont interconnectées au sein d'un réseau TCP/IP local, chaque grappe ayant un de ses ordinateurs utilisé en tant que passerelle et les autres n'étant pas atteignables depuis ce réseau. Tous ces ordinateurs fonctionnent sous GNU/Linux 2.6.30 et seulement ceux de la première grappe et l'ordinateur passerelle de la seconde grappe acceptent des commandes par SSH et des transferts de fichiers par SCP, les autres ordinateurs de la seconde ne peuvent être utilisés que par l'ordonnanceur de lots TORQUE.

Application parallèle L'application comporte quatre éléments : un lanceur, un code parallèle, une bibliothèque partagée, et un jeu de données à traiter. Le lanceur est un script `shell` qui lit un fichier de noms d'ordinateurs puis construit et soumet à distance les commandes pour le code parallèle lui-même. Ses principaux paramètres sont le nombre de processeurs sur lesquels exécuter le code parallèle, le chemin d'accès au code parallèle, et les paramètres spécifiques du code parallèle. Le code parallèle lui-même est un programme binaire GNU/Linux AMD 64 bits faisant appel à la bibliothèque partagée et dont les paramètres sont le chemin d'accès au jeu de données à traiter et le chemin d'accès au jeu de données traitées. La bibliothèque partagée elle-même est un fichier binaire GNU/Linux AMD 64 bits implantant les fonctions de communication

inter-processus et utilisant le pilote Infiniband. Enfin, le jeu de données à traiter est un fichier de 10 Go.

Conditions initiales L'ordinateur de l'utilisateur est connecté au sein du réseau local dans lequel se trouvent les ordinateurs passerelles des grappes d'ordinateurs. De même que précédemment, les fichiers de l'application parallèle se trouvent dans un sous-répertoire `etude_parallele` du répertoire personnel de l'utilisateur, sur son ordinateur. A priori, l'application est compatible avec une des grappes d'ordinateurs.

Processus de déploiement

Afin de déployer son application parallèle, l'utilisateur suit les trois étapes du processus de déploiement que sont l'allocation, l'installation, et l'exécution. Les détails de ces étapes suivent.

Allocation En analysant les besoins en ressources de son application et les capacités des ressources auxquelles il a accès, l'utilisateur déduit que son application devra utiliser la première grappe. De même que précédemment, ce choix est essentiellement dû à l'architecture des processeurs. Cependant pour s'assurer que son application fonctionnera correctement il doit également vérifier les capacités de mémorisation disponibles sur les ordinateurs et que tous les ordinateurs communiquent directement.

Installation Afin d'installer son application, l'utilisateur doit d'abord en transférer le sous-répertoire depuis son répertoire personnel sur son ordinateur vers celui sur l'ordinateur passerelle de la grappe allouée. Tous les ordinateurs de la grappe partageant les répertoires personnels, il ne doit faire ce transfert qu'une fois. Il doit ensuite produire le fichier de configuration contenant les noms de tous les ordinateurs de la grappe avant de le transférer dans le sous-répertoire de son application sur l'ordinateur passerelle. Par ailleurs, afin que la bibliothèque partagée et donc son application bénéficient du réseau Infiniband, l'utilisateur doit prévoir d'ajouter un paramètre dans la commande du lanceur de son application. Il doit déterminer le nombre de processeurs sur lequel il veut exécuter son application, un nombre inférieur au nombre total de processeurs dans la grappe. Il doit enfin découvrir le chemin d'accès au sous-répertoire de son application dans son répertoire personnel sur l'ordinateur passerelle et supposer que ce chemin sera identique sur tous les ordinateurs de la grappe. Il aura ainsi construit complètement la commande pour le lanceur de son application.

Exécution De même que pour son application paramétrique, afin d'exécuter son application parallèle, l'utilisateur doit soumettre à distance la commande du lanceur puis transférer le jeu de données traitées vers son répertoire personnel sur son ordinateur.

Bien que l'étape d'exécution soit presque aussi simple que pour une application paramétrique sur un ordinateur distant, les utilisateurs finaux peuvent rencontrer de sérieuses difficultés pour réaliser les étapes d'allocation et d'installation de leurs applications parallèles. De surcroît, dans le cas où les chemins d'accès aux répertoires personnels partagés par les ordinateurs seraient différents, alors le lanceur ne fonctionnerait pas et l'utilisateur devrait le remplacer. Cependant si ce cas de déploiement est plus complexe que le précédent, le suivant l'est davantage.

3.3.3 Application en flot de travail sur grille d'ordinateurs

Le troisième et dernier exemple est le processus de déploiement d'une application en flot de travail sur une grille d'ordinateurs distants.

Cas de déploiement

Un utilisateur veut déployer son application en flot de travail et il a accès à une grille d'ordinateurs distants. Les détails de la situation dans laquelle il se trouve suivent.

Grille d'ordinateurs Les ordinateurs auxquels l'utilisateur a accès sont une grille de stations de travail multi-processeur réparties en deux grappes. Les deux grappes sont celles décrites précédemment. Le système d'information et le gestionnaire de tâches s'exécutent sur les ordinateurs passerelles.

Application en flot de travail L'application comporte neuf éléments : un programme principal, un conteneur de composants, cinq composants a, b, c, et d, un moteur de flot de travail, une description de flot de travail, et un jeu de données à traiter. Le programme principal est un fichier binaire GNU/Linux I686 32 bits activant le composant moteur de flot de travail dans un conteneur. Le conteneur de composants est un fichier binaire GNU/Linux I686 32 bits dont le paramètre est le chemin d'accès au composant qu'il devra charger. Les composants sont des bibliothèques binaires GNU/Linux I686 32 bits partagées. Le composant moteur de flot de travail sait lire une description de flot de travail puis en charger et en connecter les composants dans des conteneurs. Le composant a sait lire un jeu de données à traiter et envoyer des données pré-traitées à deux composants b et c. Les composants b et c savent recevoir des données pré-traitées et les envoyer à un composant d. Le composant d sait écrire un jeu de données traitées. La description du flot de travail est un fichier au format XML décrivant l'assemblage en losange des composants a, b, c, et d, d'une part, et paramétrant les composants moteur de flot de travail, a, et d respectivement avec les chemins d'accès à la description du flot de travail et aux jeux de données à traiter et traitées, d'autre part. Enfin, le jeu de données à traiter est un fichier de 5 Go.

Conditions initiales De même que précédemment, l'ordinateur de l'utilisateur est connecté au sein du réseau local dans lequel se trouvent les ordinateurs passerelles des grappes d'ordinateurs de la grille. Les fichiers de l'application en flot de travail se trouvent dans un sous-répertoire `etude_flot` du répertoire personnel de l'utilisateur, sur son ordinateur. Les composants sont dans des archives au format Zip. A priori, l'application est compatible avec une des grappes d'ordinateurs de la grille.

Processus de déploiement

Afin de déployer son application en flot de travail, l'utilisateur suit les trois étapes du processus de déploiement que sont l'allocation, l'installation, et l'exécution. Les détails de ces étapes suivent.

Allocation De même que précédemment, afin d'allouer des ordinateurs pour son application, l'utilisateur doit analyser les besoins en ressources de son application et les capacités des ordinateurs de la grille. Pour ce faire, il doit notamment consulter le système d'information de la grille. Il déduit que son application devra utiliser des ordinateurs de la seconde grappe.

Installation De même que précédemment, afin d'installer son application, l'utilisateur doit en transférer le sous-répertoire depuis son répertoire personnel sur son ordinateur vers celui sur l'ordinateur passerelle de la grappe allouée. Il doit ensuite mettre à jour la description du flot de travail de son application afin que les chemins d'accès aux jeux de données soient corrects. Il doit enfin préparer la description de la tâche qu'il devra soumettre au gestionnaire de tâches de la

grille afin d'exécuter son application sur les ordinateurs qu'il a sélectionnés. Ne supportant pas les applications en flot de travail en tant que telles mais seulement des processus indépendants, ce format de description de tâche impose à l'utilisateur de décrire une tâche dans laquelle sont démarrés simultanément le programme principal et tous les conteneurs de son application.

Exécution Afin d'exécuter son application, l'utilisateur en soumet la description de tâche au gestionnaire de tâches de la grille. Celui-ci démarre le programme principal et tous les conteneurs sur les ordinateurs indiqués dans cette description. Enfin, une fois le programme principal terminé, l'utilisateur transfère le jeu de données traitées depuis son répertoire personnel sur l'ordinateur passerelle de la seconde grappe vers son répertoire personnel de son ordinateur.

3.3.4 Discussion

Dans le cas du déploiement d'une application paramétrique sur un ordinateur, on imagine facilement l'intérêt pour l'utilisateur qu'aurait un lanceur plus développé. Pour l'étape d'exécution par exemple, au lieu de construire et de soumettre localement les commandes pour les codes paramétriques, ce qui oblige l'utilisateur à utiliser le protocole SSH, un tel lanceur pourrait accepter les paramètres pour SSH et utiliser lui-même ce protocole d'action à distance. De même, un tel lanceur pourrait prendre en charge l'installation de l'application par SCP sur l'ordinateur alloué, voire l'allocation de l'ordinateur.

Non seulement un tel lanceur faciliterait la tâche de l'utilisateur, mais des optimisations de l'étape d'exécution deviendraient alors possibles. Par exemple, au lieu de soumettre tous les codes paramétriques simultanément, connaissant le nombre total de cœurs disponibles sur l'ordinateur alloué (justement suite à son allocation), un tel lanceur pourrait ne soumettre qu'un nombre suffisant de codes paramétriques pour utiliser tous les cœurs puis en soumettre de nouveaux quand les anciens auraient terminé.

De même, dans le cas du déploiement d'une application parallèle sur une grappe d'ordinateurs, on imagine facilement l'intérêt pour l'utilisateur qu'aurait un lanceur qui puisse tenir compte des chemins d'accès potentiellement différents aux répertoires personnels partagés par les ordinateurs, d'une part, et qui puisse paramétrer la bibliothèque de communication pour les technologies disponibles entre les ordinateurs, d'autre part.

Enfin, dans le cas du déploiement d'une application en flot de travail sur une grille d'ordinateurs, devoir allouer tous les ordinateurs pour toute la durée de l'exécution de l'application n'est pas optimal. En effet, l'application est justement conçue pour n'utiliser des ressources que quand ses tâches en ont besoin, exécuter les conteneurs tout au long de l'application est donc utiliser des ressources pour rien.

Dans les exemples donnés, on n'a pas tenu compte des temps d'exécution et donc des réservations d'ordinateurs à effectuer, ce qui est pourtant nécessaire dans les cas où de nombreux utilisateurs se partagent une même ressource (grappe et grille surtout). Le processus de déploiement est donc encore plus complexe pour l'utilisateur.

Enfin, on n'a pas présenté d'exemple d'application hétérogène. Par exemple, on aurait pu présenter une application couplant une application paramétrique et une application parallèle. Là encore, le déploiement est encore plus complexe pour l'utilisateur.

Les deux sections suivantes présentent l'état de l'art en matière d'allocation, d'installation, et d'exécution puis de déploiement.

3.4 Allocation, installation, et exécution

Allouer, installer, et exécuter sont des activités fréquentes en informatique. De nombreux travaux facilitent ces étapes dans des conditions particulières. Cette section présente différents travaux en rapport avec l'allocation de ressources, l'installation d'applications, et l'exécution d'applications.

3.4.1 Allocation

Allouer des ressources pour des applications requiert les trois éléments suivants : un modèle d'abstraction de ressources, d'applications, et d'allocations (entités) ; un processus d'abstraction des entités selon ce modèle ; et un processus décisionnel d'allocation. Ces trois éléments sont respectivement appelés langages de description, systèmes d'information, et ordonnanceurs.

Langages de description

Ressources Il y a peu de langages de description de ressources de calcul. Ces langages peuvent comporter les caractéristiques statiques voire dynamiques des ordinateurs. Sous la forme de paires clef-valeur, ces langages reprennent les caractéristiques fournies par les constructeurs, telles que le type, le nombre, et la fréquence des processeurs, les capacités de la mémoire vive et des disques durs, les types d'interfaces et les noms de l'ordinateur sur le réseau, et la famille et le nom de son système d'exploitation. À ces caractéristiques statiques peuvent s'ajouter des caractéristiques dynamiques, telles que l'état de fonctionnement de l'ordinateur et la charge de ses processeurs.

Publiée en mars 2009 par les membres du groupe de travail GLUE-WG du forum ouvert de la grille (*Open Grid Forum*, OGF), la version 2.0 de la spécification GLUE [50] propose un modèle abstrait d'information pour les grilles particulièrement abouti. Ce modèle distingue les domaines des administrateurs (organisations réelles) et les domaines des utilisateurs (organisations virtuelles), ces deux types d'organisation pouvant être hiérarchiques. Ces domaines peuvent avoir accès à des services de calcul ou bien de mémorisation via des points terminaux (*endpoints*) et selon des politiques d'accès. Ces services exploitent des ressources potentiellement partagées et leur technologie est précisément décrite. Ce modèle décrit également les activités et les données des domaines utilisateurs ainsi que les environnements d'exécution et de mémorisation dont ils disposent au sein de la grille.

Applications Les langages de description d'applications peuvent être regroupés en quatre catégories : les langages de soumission de tâches (*jobs*), les langages de description d'architecture logicielle, les langages de description et d'assemblage de composants logiciels, et les langages de flot de travail.

Évolutions des commandes `shell`, les langages de soumission de tâches offrent une description minimale des applications afin de réussir leur allocation. Ces langages sont notamment utilisés par des ordonnanceurs de lots, tels que ceux cités plus bas. Parmi ces langages figure la version 1.0 du langage de description de soumission de tâches (*Job Submission Description Language*, JSDL) [51] proposé fin 2005 par le forum ouvert de la grille (*Open Grid Forum*, OGF). En plus de la commande, ce langage permet de spécifier des variables d'environnement, des fichiers à placer avant et à déplacer après l'exécution, des durées totales d'exécution, et même des caractéristiques de ressources de calcul attendues. Ces caractéristiques sont spécifiées dans les termes simples des langages de description de ressources présentés ci-dessus. Par exemple, une description JSDL peut comporter la capacité minimale en mémoire vive de l'ordinateur attendu. Ces langages n'offrent qu'une description restreinte des applications.

Issus du domaine du génie logiciel, les langages de description d'architecture logicielle (*Architecture Description Languages*, ADLs) ont un autre objectif que les langages de soumission de tâches. Ils permettent de spécifier et de concevoir des applications notamment en définissant des composants et des ports, puis en associant les ports aux composants, et enfin en connectant les ports. Ces langages suivent le principe de séparation des préoccupations [52], et plus particulièrement le patron de conception d'inversion du contrôle (*Inversion of Control*, IoC) [53]. Parmi ces langages figurent Acme [54], Wright [55], et Darwin [56]. Acme a d'abord été conçu en tant que langage d'échange de description d'architecture entre applications. Il permet notamment de définir et d'analyser des styles d'architecture. Wright apporte un formalisme permettant de valider la consistence d'une architecture. Enfin, bénéficiant d'un environnement d' exécution, Darwin permet également de définir le comportement temporel d'une architecture tout en le contraignant afin d'éviter des dysfonctionnements. S'ils décrivent bien les applications, ces langages restent indépendants des implantations qu'ils décrivent.

Les langages de description et d'assemblage de composants logiciels font partie des technologies d'applications à base de composants présentées dans le chapitre précédent. Ces technologies visent à porter les bonnes pratiques du domaine du matériel vers le domaine logiciel. S'ils ne sont généralement pas liés à un langage de programmation, ces langages définissent cependant les applications en combinant des implantations des composants que celles-ci comportent. Dans son modèle de composants CORBA (*CORBA Component Model*, CCM) [27], le groupe de gestion d'objets (*Object Management Group*, OMG) propose un langage de description et d'assemblage de composants, respectivement appelés langage de description de composant logiciel (*Component Software Descriptor*, CSD), et langage de description d'assemblage de composants (*Component Assembly Descriptor*, CAD). Fractal [28] et SCA [29] ont des langages similaires. Toutefois, avoir plusieurs langages de description d'application afin d'allouer des ressources oblige de savoir les interprêter tous.

Le langage de description générique d'application (*Generic Application Description*, GADe) [57] permet de décrire des applications parallèles, réparties, et hybrides. L'objectif de GADe est de rendre possible l'allocation de ressources pour des applications de ce type en offrant un langage de description pour les technologies d'application qui n'en ont pas et en remplaçant les langages spécifiques de description des technologies d'application qui en ont. GADe permet ainsi de décrire à la fois des applications parallèles par passage de messages, des applications à base de composants, et des applications à base de composants parallèles, utilisant respectivement les technologies MPI [15], CCM [27], et GridCCM [30]. En revanche GADe ne permet pas de décrire des applications dynamiques.

Les technologies d'application en flot de travail fournissent des langages de description d'applications qui évoluent au cours du temps. Standardisé en mars 2007 par l'organisation pour l'avancement de standards d'information structurée (*Organization for Advancement of Structured Information Standards*, OASIS), la version 2.0 du langage d'exécution de procédure d'entreprise (*Business Process Execution Language*, BPEL) est un langage fondé sur le format de document XML permettant de décrire des applications en flot de travail de grande taille. Beaucoup d'autres langages de description de flots de travail existent. De même que pour les langages de description d'architecture logicielle, ces langages sont indépendants des implantations des tâches qu'ils coordonnent.

Enfin, ces quatre catégories sont perméables les unes avec les autres. Par exemple, le modèle de composants spatio-temporel (*Spatio-Temporal Component Model*, STCM) [58] est un modèle de composants qui intègre une sémantique temporelle dans les connexions entre composants. Ce modèle permet ainsi de décrire aussi bien des applications à base de composants que des applications en flot de travail.

Allocations Les langages de description d'allocation sont beaucoup plus rares que les langages de description d'application ou bien de ressources. Dans la version 2 de sa spécification du langage unifié de modélisation (*Unified Modeling Language*, UML) [59], l'OMG propose cependant un type de diagramme appelé diagramme de déploiement dont l'expression consiste à représenter des artefacts sur des nœuds, c'est-à-dire des éléments logiciels sur des éléments matériels. Dans les systèmes d'information, les éléments prenant part aux allocations sont généralement représentés par des ressources, les éléments logiciels devenant des ressources à leur tour.

Systèmes d'information

Les systèmes d'information sont des outils permettant d'obtenir de l'information à propos des ressources de calcul, des applications, et des allocations, selon leurs spécialités. Ces outils peuvent utiliser les langages de description présentés ci-dessus afin de présenter cette information. L'information qu'ils fournissent a une durée de vie limitée, c'est pourquoi ils doivent la rafraîchir régulièrement. Par exemple, le système d'exploitation GNU/Linux comporte un système d'information enregistrant dans des fichiers et en mémoire vive les états presque instantanés des ressources de calcul, du système d'exploitation, et des applications. Cette information peut être obtenue en consultant ces fichiers, en appelant des commandes, voire en utilisant l'interface de programmation du système. Pour obtenir les caractéristiques des processeurs, on peut consulter le fichier `procinfo`, pour la capacité mémoire disponible, utiliser la commande `top`, et pour le type d'un fichier, utiliser la bibliothèque `libmagic`, etc. Au niveau du système d'exploitation, l'information est brute et donc difficile à utiliser.

Des systèmes d'information pour des réseaux d'ordinateurs permettent de consolider toute l'information disponible sur chacune d'elle et d'alerter les administrateurs en cas d'incident. Par exemple, Ganglia et Nagios sont couramment utilisés afin de surveiller des parcs entiers d'ordinateurs. Grâce à un système avancé de sondes, Nagios permet de surveiller non seulement le matériel mais également certaines ressources de plus haut niveau, telles qu'un serveur HTTP. Le système d'information de la boîte à outils Globus pour la grille d'ordinateurs est appelé système de suivi et de découverte (*Monitoring and Discovery System*, MDS). Ce système fournit un mécanisme d'agrégation et de déclenchement de procédures particulièrement abouti. Le service météorologique du réseau (*Network Weather Service*, NWS) [60] est un système réparti comportant différentes sondes et des répétiteurs afin de mesurer, prévoir, et diffuser les performances des processeurs et des connexions point à point par exemple. Fondé sur un modèle multi-agent, MonALISA (*Monitoring Agents using a Large Integrated Services Architecture*) [61] est un système dynamique de services pour le suivi, le contrôle, et l'optimisation pour la grille. Ces systèmes sont principalement conçus pour surveiller et découvrir des ressources mais ils ne permettent pas d'obtenir d'information à propos des besoins en ressources des applications.

Pour obtenir davantage d'information sur les applications que les systèmes d'exploitation ne le permettent directement, des outils de profilage existent. Généralement utilisés pendant le développement des applications, ces outils permettent de surveiller la consommation en temps de calcul, en mémoire, et en communication. Par exemple, oprofile [62] est un outil permettant de tracer et d'analyser la performance des applications s'exécutant sur GNU/Linux. Valgrind [63] est un émulateur permettant d'analyser la consommation d'un programme en mémoire. tau (*Tuning and Analysis Utilities*) [64] est un outil de profilage spécialisé pour les applications parallèles. Enfin, le cadriciel d'ingénierie inverse ERESI [65] offre une approche modulaire pour instrumenter des applications et en tracer la performance. Tous ces outils sont toutefois rarement utilisés afin d'allouer des ressources pour les applications, ils sont cependant un des rares moyens d'obtenir de l'information à propos des applications.

Ordonnanceurs

Les ordonnanceurs sont les éléments actifs qui allouent des « forces de travail » à des « charges de travail ». On trouve des ordonnanceurs dans les systèmes d'exploitation voire dans certaines technologies d'application. À l'échelle de l'ordinateur, les ressources sont partagées entre plusieurs applications grâce à un ordonnanceur. Maui est un ordonnanceur pour grappe d'ordinateurs, DAGman pour les grilles Condor-G [66], et GridWay pour les grilles Globus.

L'outil de planification de l'exécution dans les grilles (*Planning for Execution in Grids*, Pegasus) [67] planifie l'exécution d'applications suivant un modèle de traitement en flots de données sur des ressources réparties. Les applications sont décrites sous forme de graphes orientés sans cycle dont les arcs sont des données, les sommets des programmes. Les ressources sont servies par l'intergiciel de grille de calcul Globus.

L'objectif de Pegasus est de réduire la durée d'exécution des flots de données en les plaçant au mieux sur les ressources. Les durées des traitements et de la circulation des données sont prises en compte. En effet, pour établir l'allocation d'une ressource pour un programme, Pegasus utilise trois types d'information :

- La description des ressources ;
- La description du lieu où sont stockées les données référencées dans les flots de données ;
- La description du lieu où sont stockés les programmes référencés dans les flots de données.

Pour réaliser une exécution planifiée, Pegasus a besoin du moteur de flots de données DAGMan et du service d'exécution Condor-G. Grâce à DAGMan, si l'exécution d'un programme échoue, alors cette exécution est rejouée.

Les ordonnanceurs doivent résoudre un problème simple mais pour lequel un grand nombre de solutions peuvent exister et dont les paramètres peuvent varier rapidement. Plusieurs optimisations provenant du domaine de la recherche opérationnelle permettent d'améliorer la qualité de ces allocations : la résolution de problème de satisfaction de contraintes, les systèmes experts, et les meta-heuristiques notamment. La fréquence d'allocation peut nécessiter la mise en œuvre de planification continue, telle que le permet le cadriciel de planification continue et d'exécution (*Continuous Planning and Execution Framework*, CPEF) [68] issu de travaux en intelligence artificielle.

3.4.2 Installation

Installer une application sur des ressources requiert toujours une application dans un état de conditionnement et un outil correspondant à ce conditionnement, déjà installé sur les ressources. Cette installation peut comporter non seulement une étape de mise en place de fichiers, mais également une étape de compilation, de configuration, ou les trois.

Avant d'être exécutée, une application est concrètement organisée en fichiers. En dehors des données qu'elle traite, les fichiers d'une application (logiciel) se regroupent dans les quatre catégories suivantes :

Code source Le code source permet à d'autres applications (suite de compilation) de produire les exécutables, les bibliothèques, et la documentation. Selon les licences choisies par les auteurs, certaines applications sont diffusées avec leur code source voire seulement par leur code source.

Exécutables et bibliothèques Les exécutables peuvent être des scripts ou des binaires. Pour être exécuté, un exécutable peut avoir besoin d'autres exécutables ou de bibliothèques et un script a besoin d'un interprêteur.

Configuration Lorsque l'application est configurable, elle acquiert sa configuration en la lisant dans des fichiers de configuration ou bien dans une base de registre du système sur lequel

elle est installée. Dans les deux cas, une partie des fichiers de l'application peut concerner la configuration, la modification de la base de registre, ou bien les deux.

Documentation La documentation peut comporter des pages de manuel, des pages HTML, des fichiers PDF, TeXinfo, HLP, etc.

Pour en faciliter la diffusion, les fichiers d'une application sont conditionnés sous forme d'archive, de paquet, ou de composant, voire d'image disque.

Images disques

Les images disques sont des copies complètes des systèmes de fichiers dans lesquels figurent les systèmes d'exploitation et toutes les applications qui y sont installées. Elles comportent généralement une partie paramétrable afin, par exemple, d'identifier chaque ordinateur. Ce type de conditionnement est fréquemment utilisé pour pré-installer industriellement des systèmes d'exploitation et des applications sur des ordinateurs homogènes, chez des constructeurs d'ordinateurs par exemple. Ces images peuvent être construites puis installées manuellement ou à l'aide d'outils, tels que SystemImager ou Kadeploy [69]. Ces outils utilisent respectivement les capacités d'amorçage par le réseau ou bien les interfaces d'administration à distance des ordinateurs. Certains outils permettent également de répliquer les modifications apportées à une image disque sur tout un ensemble d'ordinateurs, l'image originale étant appelée « image d'or ». Enfin certains outils construisent l'image disque à la volée, pendant son installation sur l'ordinateur, à partir d'un autre conditionnement, tel que des paquets. C'est par exemple le cas de l'outil d'installation complètement automatisée (*Fully Automated installation*, FAI) [70] pour le système d'exploitation Debian GNU/Linux.

Toutefois les applications elles-mêmes sont rarement conditionnées sous la forme d'images disques.

Archives

Une archive est un fichier contenant d'autres fichiers et répertoires de fichiers. Généralement compressée, une archive est facile à transmettre. GNU Tar, GNU cpio, RAR, et ZIP sont des exemples d'outils définissant des formats d'archives et permettant d'en manipuler.

Lorsque seul le code source d'une application est disponible ou bien que des optimisations supplémentaires du code binaire puissent être obtenues avec cette étape, l'installation comporte également une étape de compilation. Les outils GNU autotools et GNU Compiler Collection permettent de compiler du code source écrit dans de nombreux langages de programmation.

Paquets

Un paquet est une archive de fichiers à laquelle ont été ajoutées des meta-données. Ces meta-données sont généralement décrites dans un fichier particulier, lui-même contenu dans l'archive. Un paquet peut contenir des fichiers de toute sorte. S'il contient des fichiers de code source seulement alors un paquet est généralement qualifié de paquet source, sinon de paquet binaire. Les meta-données peuvent par exemple comporter le nom du paquet, sa description, sa version, les noms des paquets dont il dépend, une séquence de questions afin de configurer le contenu en l'installant, des traductions des meta-données, des valeurs de contrôle d'intégrité, des signatures électroniques, les noms de ses auteurs, les noms des paquets incompatibles, etc.

Les dépendances entre des paquets explicitent les dépendances entre les contenus de ces paquets. Par exemple un paquet peut contenir un fichier exécutable et dépendre d'un paquet contenant une bibliothèque partagée dont cet exécutable dépend. Les outils permettant de manipuler des paquets maintiennent généralement des bases de paquets afin de pouvoir vérifier que

sont respectées ces dépendances au moment de l'installation. Plusieurs types de dépendances peuvent exister : dépendances fortes, recommandations, etc.

Installant le contenu des paquets dans l'arborescence du système d'exploitation, la plupart des outils de manipulation de paquets requièrent d'être administrateur du système afin de les utiliser. Par exemple dans la distribution Debian GNU/Linux, un tel outil est APT/dpkg, et dans Fedora GNU/Linux, YUM/rpm. Grâce au format de configuration de paquet debconf, APT/dpkg sait configurer le code pendant l'installation. APT/dpkg exécute des scripts de pre- et post-installation, et de pre- et post-désinstallation. Certains outils de manipulation de paquets n'en installent qu'à partir des paquets sources, en les compilant à la volée. C'est notamment le cas de l'outil portage dans Gentoo GNU/Linux mais également de l'outil apt-build dans Debian GNU/Linux.

Si des outils standard sont de fait apparus avec le développement de GNU/Linux, la fondation Linux comporte un groupe de travail sur l'empaquetage (*packaging workgroup*) afin de produire un outil standard et ainsi éviter aux développeurs de devoir produire plusieurs paquets d'une même application. L'outil autopackage va dans cette direction.

Des outils de manipulation de paquets existent également pour les utilisateurs. Par exemple, la communauté du langage de programmation Python a produit l'outil easy_install manipulant des paquets appelés Python eggs entreposés dans l'index des paquets Python (*Python Package Index*, Pypi). Dans la communauté du langage ruby, un outil similaire est appelé rubygems. Ces outils permettent à l'utilisateur d'installer les fichiers où il le souhaite, dans son répertoire personnel par exemple.

Composants

En termes de conditionnement, un composant logiciel est généralement un paquet auquel ont été ajoutées les définitions d'interfaces offertes par le composant qu'il contient. Ces interfaces sont définies dans des fichiers particuliers inclus dans le paquet. Par exemple, les composants CCM sont des archives Zip comportant un fichier CSD décrivant les meta-données du code implantant le composant et les ports offerts par ce composant, ainsi qu'un fichier de configuration des propriétés du composant (s'il en avait). Un conditionnement similaire existe par exemple pour Fractal, OSGi [71], et SCA.

3.4.3 Exécution

Exécuter une application sur des ressources est l'objectif ultime des systèmes d'exploitation et donc l'objet de nombreux outils. Par exemple, les `shells`, tels que Bash ou tcsh, permettent d'exécuter des applications sur le système local, `shell` sécurisé (*Secured Shell*, SSH) sur des systèmes distants grâce à une architecture client-serveur. Afin d'exécuter des applications sur plusieurs systèmes distants, des `shells` parallèles, tels que ClusterSSH et TakTuk [72], ont été développés. Certaines technologies d'applications fournissent également des « lanceurs », tels que `mpirun` dans l'implantation MPICH de MPI.

Si des outils permettent d'exécuter des applications en direct, d'autres fonctionnent en différé. C'est le cas des gestionnaires de lots pour les grappes de calcul, tels que TORQUE, LoadLeveler, et N1 Grid Engine. CREAM [73] et GRAM sont d'autres exemples de gestionnaires de tâches fonctionnant pour la grille d'ordinateurs.

3.5 Déploiement

Cette section présente différents travaux portant non seulement sur une étape du déploiement mais également sur plusieurs voire sur le processus complet, c'est-à-dire l'allocation de ressources

ainsi que l'installation et l'exécution d'applications. Bien que leurs groupes se recouvrent partiellement, ces travaux sont regroupés selon qu'ils proviennent du domaine de l'administration de ressources, de la gestion d'applications à base de composants, ou qu'ils soient des outils voués exclusivement au déploiement.

3.5.1 Administration de ressources

Parmi les tâches qui incombent aux administrateurs peuvent figurer l'installation et l'exécution de systèmes d'exploitation sur des ordinateurs. Pour ce faire, ils peuvent utiliser les outils d'installation et d'exécution présentés ci-dessus ou bien utiliser des produits intégrant ces outils. À l'échelle d'une grappe d'ordinateurs par exemple, la suite logicielle de ressources applicatives ouvertes pour grappes (*Open Source Cluster Application Resources*, OSCAR) [74] permet de construire une image disque puis de l'installer sur les ordinateurs, le tout à travers une interface graphique. OSCAR permet également d'installer des paquets sur tous les ordinateurs simultanément. ROCKS [75] est une solution comparable pour laquelle les applications supplémentaires sont conditionnées sur disques compacts appelés « Rolls ». Conçu pour des parcs complets d'ordinateurs, GNU CfEngine [76] est également un produit intégré d'administration. Il permet de définir des classes d'ordinateurs puis de transférer des fichiers et d'exécuter des commandes simultanément sur tous les ordinateurs d'une même classe. La boîte à outils d'administration extrême de grappes (*Extreme Cluster Administration Toolkit*, xCAT) [77] et GOsa² sont d'autres exemples de solutions intégrées.

Conçu à l'Université d'Innsbruck, le cadriciel d'enregistrement, de déploiement, et d'approvisionnement d'activité de grille (*Grid Activity Registration, Deployment and Provisioning Framework*, GLARE) [78] offre un service d'information à propos d'applications Java. Une fois enregistrées leurs applications, les utilisateurs peuvent les demander sans se préoccuper de leur installation. En effet, lorsque des applications enregistrées mais pas encore installées sont demandées, ce service exécute automatiquement les descripteurs d'installation enregistrés avec elles. Il est constitué des trois parties suivantes :

- Un registre d'applications disponibles ;
- Un registre d'applications installées sur les ressources ;
- Un mécanisme pour l'enregistrement, l'exécution, et le suivi d'installations d'applications sur des ressources.

ORYA et l'intergiciel économique de grille ont des approches originales afin de traiter l'allocation des ressources. ORYA (*Open enviRonment to deploY Applications*) [79] offre un environnement permettant de décrire les besoins en logiciel des utilisateurs et de valider la compatibilité d'une application avec ces besoins, avant de l'installer via des outils spécifiques. Dans leur intergiciel économique de grille pour le déploiement d'applications [80], Joita et al. proposent une approche de type marché libre. Dans cette approche, les ressources sont comme des marchandises pour lesquelles des vendeurs sont mis en relation avec des acheteurs afin de négocier les niveaux de service pour les ressources.

Le groupe de travail sur la gestion de la description de configuration, du déploiement, et du cycle de vie (*Configuration Description, Deployment, and Lifecycle Management*, CDDLM) [81] du forum ouvert de la grille propose à la fois un langage de description de configuration de services, un service de déploiement, et un cycle de vie que doivent respecter les services. Le langage de description de configuration (*Configuration Description Language*, CDL) est celui du produit Smartfrog [82], commercialisé par Hewlett-Packard. Grâce à une syntaxe de définition d'interface, CDL permet de définir des services, lesquels peuvent faire référence à d'autres. Les dépendances entre services sont fixées dans leurs définitions. Le service de déploiement prend en paramètre une définition de service, en résoud les références, en génère un paquet, puis l'exécute selon son cycle de vie. Le cycle de vie des services est configuration, initialisation, et démarrage.

Dans sa proposition de descripteur de déploiement de solutions (*Solution Deployment Descriptor*, SDD) [83], OASIS sépare les descriptions des fichiers de l'application, appelée description de paquet, et des meta-données relatives au déploiement de ces fichiers, appelées description de déploiement. Ces deux descriptions sont des documents au format XML. La description de paquet comporte les descriptions de fichiers, appelés artefacts, qui sont organisés en hiérarchie et qui ont un type : archive zip ou paquet RPM par exemple. Une description de paquet peut comporter des descriptions de dépendances vers des ressources. Dans la description de déploiement associée, les artefacts sont soit des éléments installables, de configuration, ou bien de localisation. La description de déploiement comporte également la description des ressources impliquées dans le déploiement, appelée topologie. Cette description des ressources peut comporter les noms d'ordinateurs de même que des noms d'artefacts déjà déployés. La description de déploiement comporte la description du résultat de sa réalisation, c'est-à-dire les ressources produites par le déploiement décrit.

3.5.2 Gestion d'applications à base de composants

Dans leurs modèles, les technologies d'applications à base de composants spécifient la forme et le cycle de vie des composants. Certaines de ces technologies spécifient également le déploiement des applications. Des travaux ont pu être menés afin de gérer automatiquement ces applications. CCM et Fractal en sont des exemples. CCM a été étendu par une spécification pour le déploiement et la configuration de ses applications, extension que l'implantation OpenCCM respecte. Pour Fractal, l'implantation Proactive comporte des fonctionnalités de déploiement. Un cadriciel de déploiement a également été conçu, cadriciel sur lequel un langage spécifique au déploiement a pu être défini.

CCM

Dans les spécifications du déploiement et de la configuration d'applications réparties à base de composants (*Deployment and Configuration of Component-based Distributed Applications*, D&C) [84] publiées en avril 2006 par l'OMG, le processus de déploiement comporte les sept phases suivantes :

Préconditions Les composants sont conditionnés correctement, les dépôts de composants et les ressources sont disponibles ;

Installation Les composants sont mis à disposition via des dépôts de composants ;

Configuration Les choix des paramétrages fonctionnels des composants sont faits ;

Planification Les plans de déploiement (les « quoi », « où », « quand », et « comment ») sont construits ;

Préparation Les opérations sont mises en œuvre sur les ressources en vue du démarrage des applications ;

Démarrage Les applications sont mises à l'état d'exécution après l'instanciation, la connexion, et la configuration de leurs composants ;

Arrêt Les applications se terminent d'elles-mêmes ou bien sont arrêtées.

Trois catégories d'acteurs sont distinguées dans ce modèle :

Développement Spécificateurs de composants, développeurs de composants, assembleurs de composants, et conditionneurs de composants ;

Administation Administrateurs des ressources ;

Déploiement Administrateurs de dépôts de composants, planificateurs de déploiement, exécuteurs de déploiement.

Pour la phase de planification, les descriptions d'artefacts (le « quoi »), de « monolithes » (le « comment »), et d'instances (le « où ») sont reliées et parcourues par des gestionnaires d'exécution pouvant réaliser trois opérations : préparer, démarrer, et arrêter une application. Les plans de déploiement correspondent à des versions annotées des assemblages de composants. Un bon planificateur choisit le plus pertinent de l'ensemble des plans de déploiement valides.

La phase de démarrage des applications comporte les trois sous-phases suivantes :

1. Création d'objets capables d'instancier et d'assembler les composants, objets appelés fabriques d'assemblages ;
2. Fabrication des assemblages de composants grâce aux fabriques d'assemblages ;
3. Exécution des applications ainsi fabriquées.

Le processus de déploiement peut être automatisé par un outil tout-en-un ou par une suite d'outils.

Mis au point au sein de l'équipe-projet INRIA Jacquard à Lille, OpenCCM [85] est une implantation de CCM, portable sur plusieurs implantations de CORBA. Couplée avec l'infrastructure informatique distribuée (*Distributed Computing Infrastructure*, DCI), OpenCCM implante l'ensemble des interfaces CCM liées au déploiement spécifiées par l'OMG. En effet, cette infrastructure permet à OpenCCM de démarrer les applications, c'est-à-dire des assemblages de composants.

Une infrastructure DCI fédère un ensemble d'ordinateurs répartis en un « domaine de déploiement unifié ». Un gestionnaire d'infrastructure DCI (DCI Manager) manipule alors ces ordinateurs et les fabriques d'assemblage à déployer. Chaque ordinateur est représenté par un gestionnaire d'ordinateur (*Node Manager*) qui manipule localement le serveur de composants, les archives de composants, et qui peut fournir des meta-données. Chaque fabrique d'assemblage est représentée par un gestionnaire de fabrique d'assemblage (*Assembly Factory Manager*) qui crée ou détruit l'assemblage via les gestionnaires d'ordinateurs. Tous les gestionnaires sont manipulés par des scripts `shell` et sont enregistrés dans le service de désignation d'objets CORBA.

Fractal

Le cadriciel Proactive [86], implantant le modèle de composants Fractal, requiert deux types de descripteurs afin de gérer le déploiement de ses applications. Ceux du premier type, produits par les administrateurs des ressources, sont appelés descripteurs de déploiement. Les descripteurs du second type, produits par les développeurs d'applications, sont appelés descripteurs d'application. Les descripteurs de déploiement comportent les descriptions des ressources, en termes de noms d'ordinateurs et de capacité d'accueil de processus ou de machines virtuelles Java. Les descriptions d'applications comportent les besoins en ressources des applications et la description de l'application elle-même en termes de fichiers exécutables ou bien de composants Proactive. Les descripteurs d'application font référence à des descripteurs de déploiement. Quand une application lui demande des ressources, le cadriciel utilise alors le descripteur de cette application et les descripteurs de déploiement auxquels il fait référence afin de choisir ces ressources et d'y exécuter l'application.

Jade [87] est un cadriciel d'administration autonome d'applications patrimoniales en multi-tiers sur des grappes d'ordinateurs. Ce cadriciel comporte trois parties : des gestionnaires d'autonomie, des services utiles à ces gestionnaires, et les emballages (*wrappers*) des applications gérées. Les éléments de ces parties sont des composants Fractal de sorte que les systèmes autonomes que Jade administre sont spécifiés avec l'ADL Fractal. Les gestionnaires implantent des boucles de contrôle, c'est-à-dire qu'ils suivent des valeurs de paramètres et peuvent réagir, par exemple en installant une application sur un ordinateur récemment ajouté à la grappe. Pour

réaliser de telles opérations, ces gestionnaires utilisent les services. Les emballages d'applications implantent une interface de gestion de cycle de vie pour les technologies particulières et ces emballages sont conditionnés, avec les applications elles-mêmes, sous forme de composants OSGi. Une variante du langage de modélisation unifié (UML), appelée Tune [88], a également été conçue afin de faciliter la conception de ces emballages.

Également fondé sur le modèle de composants Fractal, DeployWare/FDF [89] comporte les trois parties suivantes :

DeployWare Le déploiciel est un langage spécifique au domaine du déploiement. Ce langage de description est fondé sur un meta-modèle en deux parties. La première partie, appelée expert de technologie, permet aux développeurs de décrire les « personnalités » des applications à déployer. La personnalité d'une application comporte par exemple ses fichiers, ses dépendances vers d'autres applications, et les procédures pour son déploiement et son repliement, exprimées en instructions de base. Ces procédures correspondent à l'installation, la configuration, le démarrage, la gestion, l'arrêt, et la désinstallation de l'instance de personnalité, c'est-à-dire l'application déployée. La seconde partie du meta-modèle, appelée support administratif, permet aux administrateurs ou aux utilisateurs à la fois de décrire les ordinateurs et les instances de personnalités qu'ils contiennent déjà, et également de décrire le déploiement d'une (nouvelle) instance de personnalité sur des ressources. Cette dernière instance est l'application à déployer ;

Fractal Deployment Framework (FDF) Le cadriciel de déploiement Fractal est une machine virtuelle interprêtant les descriptions et capable d'exécuter des instructions de base pour différentes technologies de machines physiques. Dans le cadriciel de déploiement Fractal, à tout élément impliqué dans le déploiement, logiciel ou matériel, correspond un composant Fractal. Pendant qu'elle les interprête, cette machine valide les descriptions, en déduit deux couches de composants Fractal, et connecte ces composants. La validation est possible grâce au meta-modèle et consiste par exemple à vérifier que les dépendances logicielles sont respectées, ou à assurer qu'à chaque procédure de déploiement correspond une procédure de repliement. Dans la première couche de composants, à chaque instance de personnalité correspond un composant composite, dont les composants sont les deux suivants : un de base appelé propriétés pour les paramètres de configuration, et l'autre composite appelé dépendances dont les composants sont les instances de personnalités en dépendance ; dans la seconde couche, les procédures sont d'autres composants composites dont les composants de base sont des instructions pour les technologies de machines. La machine virtuelle connecte les composants d'une même couche selon sa hiérarchie, et les composants des instances de personnalités avec les procédures selon les ordinateurs alloués. Le déploiement d'un composant étant supporté, le déploiement de ses dépendances l'est également, par récursivité ;

DeployWare eXplorer L'explorateur déploiciel est une interface graphique de gestion de l'application déployée sur les ressources. Cette interface permet aux administrateurs de parcourir les dépendances entre logiciels et d'exécuter leurs procédures de déploiement ou de repliement.

3.5.3 Outils de déploiement

GoDIET

GoDIET [90] est un outil permettant d'installer et d'exécuter la plate-forme applicative DIET sur des ressources. Pour ce faire, ses paramètres sont une description des ressources sous forme d'un document XML et les paramètres de la plate-forme elle-même, c'est-à-dire les paramètres de

la hiérarchie des agents DIET. S'appuyant sur SCP et SSH, GoDIET transfère les fichiers de la plate-forme sur les ressources puis exécute les différents agents. Afin d'optimiser les paramètres de la plate-forme, un planificateur spécifique appelé ADePT (*Automatic Deployment Planning Tool*) [91] a également été développé.

Adage

Adage est un outil de déploiement automatique [92] mis au point au sein de l'équipe-projet INRIA PARIS, à l'IRISA de Rennes. Il analyse, via leurs descriptions, les besoins en ressources d'une application et les capacités des ressources, puis planifie et enfin réalise l'installation et l'exécution de l'application sur les ressources.

Pendant la planification de l'installation et de l'exécution, Adage convertit une description de l'application spécifique à la technologie qu'elle utilise en une description générique, ce qui permet de séparer les implantations des planificateurs et les technologies d'applications. Ce modèle de description, appelé GADe, permet de décrire aussi bien des applications réparties que des parallèles, voire des hybrides. Pour la description des ressources, Adage utilise un format XML inspiré du format produit par le MDS Globus, enrichi de l'information sur leur topologie.

Pendant la réalisation du plan, les opérations de transfert de fichiers et de soumission de commandes sont des scripts transférés sur les ressources, appelés satellites, prenant l'information du plan en paramètre. Ces satellites permettent à Adage de supporter plusieurs technologies d'applications sur plusieurs technologies de ressources.

Voici un exemple de déploiement automatisé avec Adage. L'application est à base de composants CCM utilisant l'implantation MicoCCM et les ressources sont celles de la plate-forme expérimentale de grille Grid'5000 [93].

Analyse des besoins en ressources de l'application L'utilisateur dispose d'une version exécutable de son application CCM. Via les fichiers CAD et CSD, présents dans le conditionnement de l'application, et via une fonction de conversion pour CCM, Adage produit une description GADe de l'application.

Analyse des capacités des ressources L'utilisateur dispose d'un accès aux ressources Grid'5000 via les gestionnaire OAR [94]. Il interroge le service d'information d'OAR afin de produire un fichier de description des ressources. En lisant ce fichier, Adage sait analyser les capacités des ressources.

Allocation de ressources pour l'application Adage établit l'association entre les éléments de l'application et des ressources, en fonction des capacités des ressources à répondre aux besoins de l'application.

Installation de l'application sur les ressources Via des satellites pour CCM, Adage installe l'application sur les ressources. En fonction de l'allocation, il produit les fichiers et variables de configuration de ces satellites puis transfert les fichiers exécutables, fichiers de configuration, et données de l'application en utilisant les services de mémorisation des ressources, tels que SCP.

Exécution de l'application sur les ressources Via les mêmes satellites, Adage active l'application sur les ressources. En fonction de l'allocation, il démarre le service de désignation CORBA, démarre les conteneurs de composants, charge les composants dans les conteneurs, instancie, configure, et connecte les composants.

CORDAGE

CORDAGE (*Co-deployment and Re-deployment tool based on Adage*) [95] permet de gérer plusieurs instances d'Adage afin de co-déployer et re-déployer des applications. Le noyau de

CORDAGE maintient une description interne des applications et des ressources. Il peut être télécommandé par un humain ou par une application grâce à des appels de procédures à distance (*Remote procedure Call*, RPC). Il comporte notamment une fonction de traduction de descriptions externes vers les descriptions internes.

Dans leurs descriptions internes, les applications sont des arbres dont les feuilles sont des entités paramétrées et dont les nœuds définissent une hiérarchie de groupes d'entités paramétrées. De même, les ressources sont les feuilles d'un arbre regroupées par les nœuds de cet arbre. Par convention, chaque entité a besoin d'au plus une ressource, il est donc possible à la fois d'associer des entités et des ressources, et de respecter les affinités des entités entre elles, d'une part, et celles des ressources entre elles, d'autre part. Parmi les paramètres d'une entité figure une date de fin de vie, utile pour les réservations de ressources.

Une fois ces arbres disponibles, respectivement qualifiés de logiques et de physiques, CORDAGE essaie de trouver les associations de groupes d'entités paramétrées et de ressources. Pour ce faire, il doit réserver des ressources, auprès des services de réservation des différents groupes de ressources, tant que toutes les réservations ne sont pas synchrones. Une fois qu'elles le sont, CORDAGE transmet les descriptions internes et les associations de groupes logiques et physiques à Adage qui, dès lors, peut planifier et réaliser l'installation et l'exécution de l'application sur les ressources réservées.

Afin de gérer le déploiement de plusieurs applications ou une application en plusieurs sous-applications, CORDAGE ajoute ou retire un sous-arbre dans la description interne. La démarche de réservation puis de déploiement avec Adage est similaire.

3.6 Discussion

Les langages de description de ressources et d'applications permettent fréquemment de décrire les capacités et les besoins de celles-ci en termes de calculs, mais ne le permettent que rarement en termes de données, et presque jamais en termes de communications. Dans les ordonnanceurs fournis par les technologies d'exploitation des ressources, on ne peut généralement pas décrire le fait que plusieurs ordinateurs partagent un même serveur de données. De même, il est rarement possible d'indiquer les débits et les latences des communications entre les processeurs et les mémoires vives et disques durs (intra-ordinateur), d'une part, et entre les ordinateurs (inter-ordinateur), d'autre part. Or, pour des applications qui consomment ou produisent beaucoup de données, ou pour des applications parallèles dont les processus intercommuniquent beaucoup, la prise en compte de cette information au moment de l'allocation influence les performances de ces applications.

Certains langages de description permettent de décrire les évolutions dans le temps d'applications, mais les ordonnanceurs fournis par les technologies d'exploitation de ressources ne les supportent pas et ne permettent pas de faire évoluer les allocations de ressources. Ces ordonnanceurs ne permettent généralement pas de réserver des ordinateurs à l'avance. Il est également impossible, par exemple, de libérer un ordinateur qu'une application n'utiliserait plus, ni d'en allouer un supplémentaire pour une application qui en aurait finalement besoin. Or, pour des applications en flots de travail, la prise en compte de besoins variables au moment de l'allocation influence les performances de ces applications.

Les ordonnanceurs supposent fréquemment que l'application soit installée sur toutes les ressources. Ils ne prennent jamais en compte les besoins en termes de calculs, de données, et de communications qu'induit l'installation de l'application sur les ressources. Certains langages de description de ressources permettent de décrire les applications installées, les considérant en tant que des ressources elles-mêmes. En revanche, les ordonnanceurs ne différencient pas les ressources sur lesquelles l'application serait déjà installée des autres. Or, pour des applications dont

l'installation est coûteuse, le fait que ces ordonnanceurs n'en tiennent pas compte est génant pour la faisabilité et la performance de cette installation.

Certaines technologies d'applications supportent elles-mêmes l'allocation de ressources pour leurs applications. En revanche, leurs langages de description de ressources et d'applications ainsi que leurs ordonnanceurs leur sont spécifiques et ces langages ne sont pas interopérables. Dans la plupart des technologies d'application en flots de travail par exemple, les tâches sont des boîtes noires pour lesquelles allouer des ressources ne dépend que des états d'exécution des tâches qui les précèdent dans leurs flots respectifs. Or, l'allocation doit être homogène, que des ressources et des applications soient hétérogènes ou non.

Afin d'installer une application sur des ressources, la compilation est en quelque sorte le conditionnement ultime d'un code pour des ressources particulières. Cependant compiler est en soi un processus complexe, nécessitant de nombreuses dépendances au premier rang desquelles figure un compilateur. Le code source des applications n'étant pas toujours disponible, les technologies de conditionnement de codes déjà compilés sont généralement préférées. Toutes ces technologies nécessitent au moins un gestionnaire de conditionnement.

Parmi les technologies de conditionnement, les gestionnaires d'images disques sont particulièrement lourds car construire des images disques consiste à construire un système d'exploitation entier, et ces gestionnaires nécessitent les droits absolus sur les ordinateurs ciblés. Les gestionnaires d'archives sont limités pour le conditionnement d'applications car les archives ne contiennent aucune meta-données nécessaires par exemple afin de spécifier les dépendances. Les gestionnaires de paquets comblent cette lacune mais contrairement aux gestionnaires d'archives, ils requièrent généralement des privilèges d'administrateur afin d'écrire n'importe où dans l'arborescence des systèmes d'exploitation. Ces gestionnaires sont mutuellement exclusifs en cela que si plusieurs d'entre eux devaient coexister dans un même système, ils ne sauraient échanger l'information concernant les dépendances entre les paquets qu'ils géreraient.

En tant que conditionnement, les composants logiciels ont l'avantage de ne nécessiter aucun privilège particulier pour être chargés, paramétrés, connectés, déconnectés, et déchargés selon leurs modèles. De même que pour les gestionnaires de paquets, les implantations de modèles de composants ne sont pas interopérables. Les implantations de composants de base peuvent elles-mêmes avoir des dépendances vers des codes externes, dépendances que les descriptions de composants n'explicitent pas.

Exécuter une application peut nécessiter de soumettre des commandes en séquence ou en parallèle tout en suivant de nombreux paramètres. Si les `shells` parallèles peuvent remplacer les `shells` ordinaires, peu de produits permettent d'exécuter des flots de commandes ainsi que le feraient les moteurs des technologies d'applications en flots de travail.

Des travaux provenant de différents domaines ont intégré les deux dernières voire les trois étapes du déploiement, voir la table 3.2 page 67. Les produits issus de l'administration de ressources sont généralement très intrusifs par rapport aux ressources elles-mêmes. Bien qu'ils requièrent que leur implantation soit installée sur les ressources, les produits issus de la gestion d'applications à base de composants sont plus légers que les produits d'administration des ressources. En revanche, toutes les applications doivent être conditionnées sous la forme de composants.

Parmi les travaux provenant du domaine de l'administration de ressources, aucun ne supporte l'allocation. OSCAR permet de construire et de faire évoluer des images disques pour les ordinateurs d'une grappe mais cette technologie est réservée aux administrateurs. CfEngine et Smartfrog, qui ont les mêmes limites, requièrent des serveurs spécifiques sur chaque ordinateur et sont difficiles à programmer. Comme les produits précédents, GLARE ne permet pas de décrire finement les ressources et les applications afin d'allouer dans une perspective de performance. GLARE ne supporte que les applications utilisant une machine virtuelle Java. ORYA

Proposition	Allocation	Installation	Exécution
OSCAR	non	oui	oui
CfEngine	non	oui	oui
GLARE	non	oui	oui
ORYA	non	oui	oui
Smartfrog	non	oui	oui
SDD	non	oui	oui
OpenCCM	non	oui	oui
Proactive	non	oui	oui
Jade	oui	oui	oui
DeployWare/FDF	non	oui	oui
GoDIET	oui	oui	oui
Adage	oui	oui	oui
CORDAGE	oui	oui	oui

TABLE 3.2 – Étapes du déploiement supportées par les produits existants

Proposition	Parallélisme	Hétérogénéité	Dynamicité
Jade	oui	applications	oui
GoDIET	oui	non	non
Adage	oui	oui	non
CORDAGE	oui	oui	applications

TABLE 3.3 – Propriétés supportées par les produits existants supportant toutes les étapes du déploiement

est un environnement également réparti sur toute une organisation mais son rôle est restreint à la résolution des dépendances logicielles.

Du côté des gestionnaires d'applications à base de composants, les spécifications du déploiement et de la configuration (D&C) de l'OMG proposent un modèle de conditionnement, de mémorisation, de description, d'assemblage, et de chargement/déchargement de composants CCM. Cependant ce modèle ne spécifie pas comment les conteneurs, et par eux les ressources, sont alloués aux composants. Bien que les implantations de CCM sachent charger et connecter des composants sur demande, le chargement/déchargement d'un assemblage dans MDA D&C ne peut pas se faire par partie. Le modèle de déploiement ne supporte donc pas les applications dynamiques.

Issus de la technologie d'applications à base de composants Fractal, Proactive et DeployWare/FDF ne traitent pas l'allocation de ressources aux composants. En revanche, grâce à son meta-modèle, DeployWare/FDF permet de valider la faisabilité d'une installation à partir de l'information à propos des ressources et de l'application décrites dans son langage de description. Grâce à sa boucle de contrôle, Jade permet d'allouer les ordinateurs d'une grappe aux applications qu'il déploie. Afin de réussir l'exécution des applications, c'est-à-dire de respecter l'ordre de soumission des commandes dont elles ont besoin, ces produits utilisent la hiérarchie du modèle de composants Fractal.

Jade, Adage, et CORDAGE permettent de déployer complètement des applications sur des ressources, sous certaines conditions, voir le tableau 3.3 page 67. Jade fonctionne en environnement contrôlé, c'est-à-dire qu'il détecte l'ajout d'ordinateurs dans la grappe qu'il administre, et qu'il peut réagir en les allouant aux applications sans vérifier que la compatibilité et la faisabilité de l'installation et de l'exécution de ces applications soient respectées : elles le sont par

convention. De surcroît, afin d'être administrées par Jade, les applications doivent être à la fois conditionnées sous forme de composants OSGi pour l'installation et de composants Fractal pour l'exécution.

En utilisant le modèle de description générique d'applications GADe, Adage n'a pas besoin que les applications soient particulièrement conditionnées. Le modèle de description des ressources permet de décrire les topologies des ordinateurs mais supporte mal les volumes partagés de mémorisation. GADe et ce format de description de ressources étant statiques, Adage est limité au déploiement voire au re-déploiement d'applications statiques sur des ressources statiques. Les traductions de descriptions spécifiques donnant des descriptions génériques entières, Adage ne supporte pas directement les applications utilisant simultanément plusieurs technologies. Adage ne tient pas compte des applications installées sur les ressources, qu'il s'agisse de dépendances ou de l'application à déployer. Enfin, contrairement aux ordonnanceurs de lots, Adage ne suppose évidemment pas que l'application soit déjà installée. Toutefois, lors de la planification de l'installation et de l'exécution d'une application, Adage ne prend pas en compte les besoins induits par son propre déploiement.

CORDAGE utilise Adage et hérite donc de la restriction de celui-ci aux ressources et applications statiques. Il compense toutefois cette restriction en utilisant simultanément plusieurs instances d'Adage. Pour ce faire, CORDAGE maintient une seconde description de l'application et des ressources, plus simple en termes d'information permettant de valider la compatibilité, mais qui permet de tenter de co-allouer des ordinateurs sur plusieurs grappes. Les entités de cette seconde description sont déconnectées, ce qui réduit les interactions des instances d'Adage. En pratique, cette description ne permet pas de déployer des applications réellement dynamiques, telles que des applications en flots de travail.

3.7 Conclusion

Afin de fonctionner sur les ressources, les applications conditionnées doivent y être déployées. Les administrateurs connaissent bien les ressources qu'ils administrent et les développeurs les applications qu'ils développent. Dans le contexte de l'informatique à haute performance cependant, les administrateurs s'occupent des systèmes d'exploitation et de la maintenance des ressources, et les développeurs n'ont pas d'accès privilégié aux ressources. Les applications sont donc généralement déployées par les utilisateurs eux-mêmes. Ils peuvent traiter séparément et intégrer manuellement les étapes du déploiement que sont l'allocation, l'installation, et l'exécution, mais les propriétés des ressources et des applications que sont le parallélisme, l'hétérogénéité, et la dynamicité complexifient un tel processus.

L'étape d'allocation est particulièrement délicate afin de déployer correctement une application. Quand les technologies d'applications fournissent des ordonnanceurs, ceux-ci peuvent compenser les lacunes des ordonnanceurs fournis par les technologies d'exploitation de ressources. En revanche, ils sont rarement suffisamment génériques pour laisser aux utilisateurs la possibilité de les utiliser pour toutes leurs applications, pour les applications hétérogènes en particulier. De plus, afin de faire leurs choix, ces ordonnanceurs ne considèrent pas les besoins en données, en calculs, et en communications induits par l'installation de l'application sur les ressources. Les manières d'installer une application sont nombreuses, ce qui peut expliquer pourquoi ils n'en considèrent aucune et supposent seulement les applications déjà installées.

Des travaux permettent, dans certaines conditions, d'intégrer les étapes d'installation et d'exécution voire d'allocation. Certains outils d'administration, tels que Jade, de gestion d'applications, tels que DeployWare/FDF, et de déploiement, tels qu'Adage et CORDAGE, simplifient ainsi le déploiement incombant aux utilisateurs. En revanche, les utilisateurs n'ont par définition pas le privilège d'utiliser des outils d'administration, les gestionnaires d'applications sont sou-

vent à la fois spécifiques à une technologie et mutuellement exclusifs, et les outils de déploiement ont des langages de description des ressources et des applications incomplets par rapport aux problèmes auxquels ils doivent pourtant faire face. Un modèle de déploiement intégrant les trois étapes et supportant les trois propriétés des ressources et des applications à la fois est donc à inventer.

Troisième partie

Contribution

Chapitre 4

Architecture ODD/SAMURAAIE de déploiement dynamique d'application

4.1 Introduction

Dans le contexte de l'informatique à haute performance, les contraintes du processus de déploiement sont fortes. Il est difficile voire impossible pour l'utilisateur de respecter la compatibilité de ressources avec des applications, d'une part, et la faisabilité de l'installation et de l'exécution d'applications sur des ressources, d'autre part. En effet, le nombre de ressources, de systèmes d'exploitation, et de technologies de ressources et de systèmes d'exploitation que comporte une infrastructure informatique à haute performance est à la fois grand et variable. Les applications ne sont pas moins complexes que ces infrastructures.

Ce chapitre propose une architecture de déploiement dynamique afin d'automatiser le processus d'allocation de ressources, d'installation d'applications sur les ressources allouées, et d'exécution de ces applications installées dans le contexte de l'informatique à haute performance. Il donne d'abord les principes de cette architecture. Puis il en présente le modèle d'abstraction, inspiré des principes de la classification et du déploiement présentés dans les chapitres précédents. Il en présente ensuite le modèle de déploiement sur demande, reposant sur ce modèle d'abstraction. Il discute enfin cette proposition.

4.2 Principes

Afin d'automatiser le déploiement d'applications sur des ressources, l'architecture que propose ce chapitre suit deux principes complémentaires. Le premier est un principe de modélisation, qui sépare l'environnement de déploiement, son abstraction, et sa gestion. Le second est un principe de permanence, qui relie les changements de l'environnement de déploiement à ceux de son abstraction, d'une part, et les changements de l'abstraction de l'environnement de déploiement à ceux de sa gestion, d'autre part.

4.2.1 Modélisation

L'architecture de déploiement dynamique ODD/SAMURAAIE comporte trois niveaux : l'environnement de déploiement, son abstraction, et sa gestion. L'environnement de déploiement est l'environnement de l'utilisateur. Il comporte les éléments informatiques auxquels celui-ci a accès. Il s'agit de ressources, d'applications, et de systèmes d'exploitation voire d'outils de déploiement. L'abstraction de cet environnement en est une représentation. Elle comporte l'information nécessaire à la gestion de cet environnement. Cette information porte sur les éléments informatiques

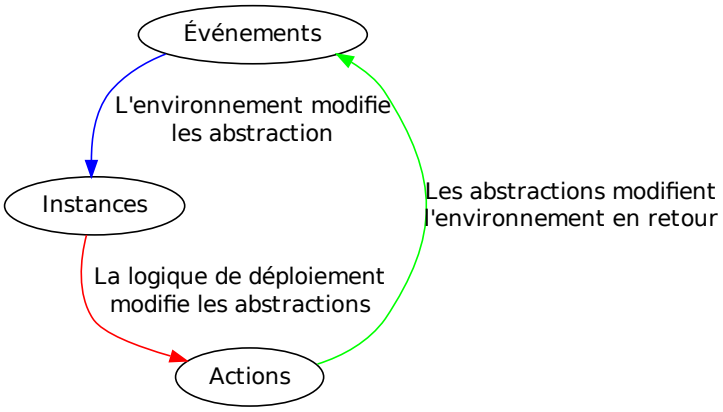


FIGURE 4.1 – Cycle permanent du déploiement

mais également sur leurs changements passés, actuels, ou prévus ainsi que sur les actions ou événements qui les ont fait, font, ou feront changer. La gestion de cet environnement est la finalité de l'architecture. Elle comporte différents mécanismes afin de déployer des applications sur des ressources et de réagir aux changements de celles-ci.

4.2.2 Permanence

L'architecture ODD/SAMURAAIE de déploiement dynamique est permanente, voir la figure 4.1 page 74. Elle est un artefact, tel que les a définis Herbert Simon dans son ouvrage « Les sciences de l'artificiel » [96]. Elle comporte une « intelligence » propre et interagit avec son environnement par son interface. Cette interface lui permet à la fois de capter et de produire des changements dans son environnement. Cette permanence peut être décomposée telle un cycle comportant trois étapes. Dans la première, un événement, qui se produit dans l'environnement, en modifie la représentation, c'est-à-dire l'abstraction (acquisition). Dans la seconde étape, la gestion réagit en modifiant l'abstraction à son tour (modification). Enfin cette dernière modification est répercutée dans l'environnement sous la forme d'une action (restitution). Ce cycle permanent rend le processus de déploiement dynamique.

Les deux précédents chapitres ont classifiés les éléments informatiques. La section suivante présente la nature et la structure de l'abstraction, appelées modèle d'abstraction SAMURAAIE, et la suivante celles des mécanismes manipulant cette abstraction, appelées modèle de déploiement sur demande ODD.

4.3 Modèle SAMURAAIE d'abstraction des systèmes informatiques

SAMURAAIE est un modèle d'abstraction des systèmes informatiques. Il est un langage de représentation de systèmes informatiques, indépendant de tout langage de représentation qui serait lié à une technologie particulière. Ce langage permet de recueillir de l'information, par définition intentionnelle et périssable, à propos de systèmes informatiques quelconques et en vue de leur gestion. En particulier, il permet de représenter l'environnement de déploiement, du point de vue de l'utilisateur, comportant des ressources, des systèmes d'exploitation, des appli-

cations, voire des outils de déploiement auxquels cet utilisateur aurait accès. Cet environnement changeant dans le temps, ce langage permet de représenter autant les éléments informatiques que les actions sur ces éléments et les événements en provenance de ces éléments, événements qui les ont fait, font, ou feront changer. Ces caractéristiques justifient le nom : *System Abstraction Model for User, Resources, and Applications (Actions on, Instances of, and Events from)* (SAMURAAIE).

Cette section présente d'abord la structure générale de SAMURAAIE que sont les graphes orientés de contenant, de contenu, et d'association. Puis elle présente les trois niveaux d'information dans SAMURAAIE que sont les instances, les actions, et les événements. Elle fournit ensuite une table de symboles permettant de figurer des abstractions SAMURAAIE. Elle propose des exemples illustrant SAMURAAIE. Enfin, elle discute ce modèle d'abstraction.

4.3.1 Contenant, contenu, et association

SAMURAAIE permet d'abstraire trois types de contraintes. Les deux premiers correspondent aux contraintes du processus de déploiement introduites dans le chapitre précédent, c'est-à-dire les contraintes de compatibilité et de faisabilité. Le troisième type de contraintes correspond à la consistance de l'abstraction avec l'environnement de déploiement. Afin d'abstraire ces trois types de contraintes, SAMURAAIE utilise la métaphore des contenants et des contenus dans laquelle l'association de contenants et de contenus n'est possible que dans des conditions particulières.

Les contraintes de compatibilité, de faisabilité, et de consistance portent autant sur des éléments informatiques que sur leurs relations. Pour la compatibilité entre des ressources et une application par exemple, des ordinateurs parmi les ressources peuvent être compatibles avec les processus de l'application, mais des ressources ne sont compatibles avec cette application que si le réseau de ces ordinateurs est compatible avec les communications inter-processus de l'application. Afin d'abstraire autant les éléments informatiques que leurs relations, SAMURAAIE utilise des graphes orientés dans lesquels des sommets peuvent être connectés par des arcs.

Éléments

Les sommets des graphes orientés représentent les éléments informatiques. Un sommet a un identifiant unique, un état de fonctionnement, et une durée de validité. Un sommet est un sommet de contenant ou bien de contenu, et simple ou bien d'agrégation. Par exemple, une mémoire est abstraite par un sommet de contenant, un fichier par un sommet de contenu. Un sommet d'agrégation agrège un ou plusieurs sommets simples ou bien d'agrégation. Par exemple, un ordinateur agrégeant une mémoire, un processeur, et des connecteurs est abstrait par un sommet d'agrégation ordinateur, sa mémoire, son processeur, et ses connecteurs respectivement par des sommets simples mémoire, processeur, et connecteurs.

L'état d'un sommet peut être un des états suivants (voir la figure 4.2 page 76) :

`ENREGISTRÉ` L'élément informatique est abstrait mais son état de fonctionnement est inconnu ;

`ATTENDU` L'élément informatique est attendu ou va fonctionner correctement ;

`CORROMPU` Quelque chose de « négatif » est arrivé à l'élément informatique ;

`OK` L'élément informatique fonctionne correctement ou est en cours de modification ;

`PROSCRIT` L'élément informatique ne devrait plus fonctionner correctement ou ne devrait plus être modifié ;

`SUSPENDU` L'élément informatique est connu mais il ne fonctionne pas encore, ne fonctionne plus, n'est pas encore en cours de modification, ou n'est plus en cours de modification ;

`ARRÊTÉ` L'élément informatique ou sa modification ont été arrêtés alors qu'ils fonctionnaient, avant qu'ils ne se terminent ;

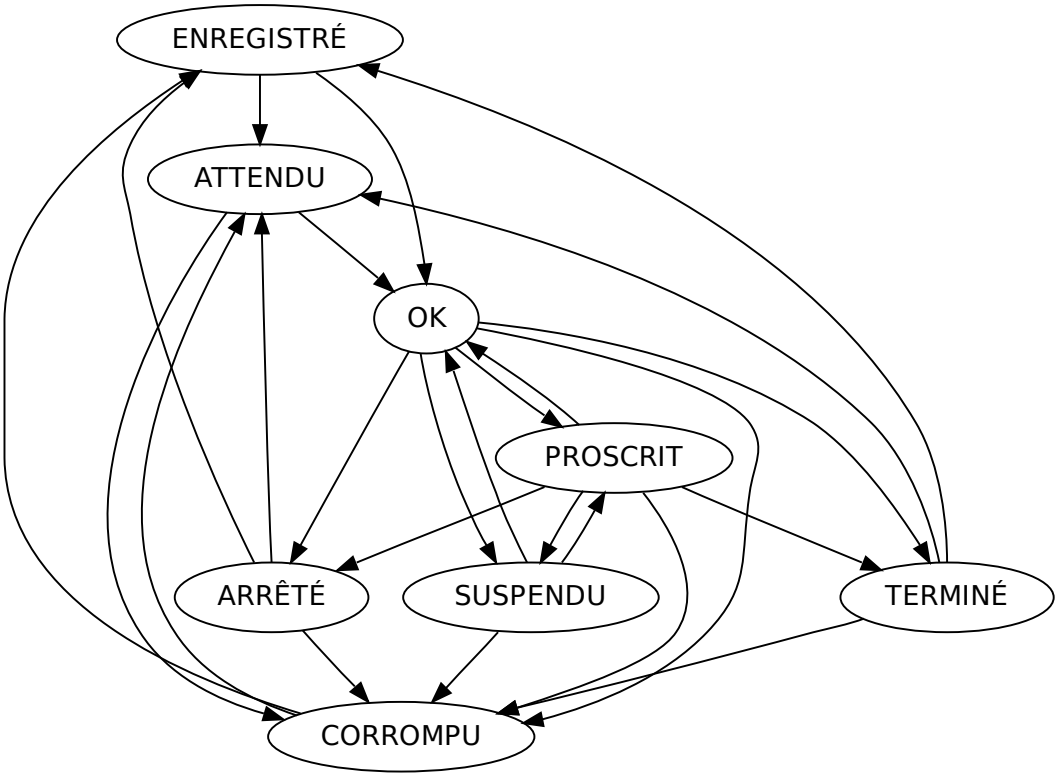


FIGURE 4.2 – États possibles d'un sommet

`TERMINÉ` L'élément informatique ou sa modification a terminé et ne fonctionnent plus.

Les durées de validité de l'information portée par des sommets peuvent être spécifiées de deux manières. La première est de spécifier deux dates complètes, la seconde de spécifier une quantité de temps. Les intervalles spécifiés par les dates sont ordonnés directement. Pour les quantités de temps, elles peuvent être ordonnées par des arcs de précédence entre les sommets qu'elles spécifient.

Relations

Les arcs des graphes orientés représentent les relations entre les éléments informatiques. Ils ont un type et deux sommets incidents, en source et en destination. Parmi tous les types d'arcs figurent la liaison, l'agrégation, et la précédence. Les arcs de liaison lient des sommets de contenant à des sommets de contenu. Par exemple, l'allocation d'une mémoire à un fichier est abstraite par un arc de liaison depuis le sommet mémoire vers le sommet fichier. Les arcs d'agrégation agrègent des sommets en d'autres. Par exemple, l'agrégation d'une mémoire, d'un processeur, et de connecteurs en un ordinateur est abstraite par des arcs d'agrégation respectivement depuis les sommets mémoire, processeur, et connecteurs vers le sommet ordinateur. Les arcs de précédence ordonnent des sommets dans le temps. Par exemple, une séquence de processus est abstraite par un arc de précédence entre les sommets du premier et du second processus.

L'ensemble des sommets de contenant et des arcs qui les relient est appelé graphe de contenant, celui des sommets de contenu et des arcs qui les relient graphe de contenu, et celui des sommets de contenant et de contenu reliés par des arcs de liaison et les arcs de liaison eux-mêmes le graphe d'association. Ces graphes sont spécialisés selon leur niveau d'abstraction : graphes d'instances, d'actions, ou d'événements.

4.3.2 Instances, actions, et événements

Afin de gérer le déploiement d'applications sur des ressources dans le contexte de l'informatique à haute performance, SAMURAAIE permet d'abstraire les instances de ressources et d'applications, les actions sur les ressources pour les applications, et les événements portant sur ces instances et ces actions. À ces trois niveaux d'abstraction correspondent les trois types de contraintes introduites ci-dessus, qui peuvent également être qualifiées de contraintes architecturales, logistiques, et de consistance. À chacun de ces trois niveaux correspondent trois graphes orientés, ou abstractions : les graphes de contenant, de contenu, et d'association. Ce découpage donne neuf abstractions dont les noms figurent dans le tableau 4.2 page 78.

Ressources, Application, et Carte

Le premier niveau d'abstraction, le niveau Instances, comporte les abstractions Ressources, Application, et leur éventuelle association, Carte. Les sommets de ces abstractions spécialisent les sommets de base, et les abstractions Ressources et Application introduisent de nouveaux types d'arcs.

Éléments d'instance Les types de sommets des abstractions Ressources et Application figurent le découpage en information, opération, et transmission introduit pour la classification présentée dans les deux chapitres précédents. Ils figurent également les caractéristiques de ces trois sous-classes.

Les ressources comportent des ordinateurs, éventuellement interconnectés, c'est-à-dire des mémoires, des processeurs, et des connecteurs. L'abstraction Ressources comporte donc des sommets d'agrégation Ordinateur et des sommets simples Mémoire, Processeur, et Connecteur.

(a) Symboles des éléments SAMURAAIE

Élément	Profil	Style
Ordinateur	Ellipse	Vide
Mémoire	Rectangle	Vide
Processeur	Losange	Vide
Connecteur	Cercle	Vide
Processus	Ellipse	Plein
Blob	Rectangle	Plein
Activité	Losange	Plein
Flux	Cercle	Plein
Système	Ellipse	Vide
Service de mémorisation	Rectangle	Vide
Service d'exécution	Losange	Vide
Service d'adressage	Cercle	Vide
Tâche	Ellipse	Plein
Requête de mémorisation	Rectangle	Plein
Requête d'exécution	Losange	Plein
Requête d'adressage	Cercle	Plein
Zone	Ellipse	Vide
Capteur	Triangle	Vide
Modification	Ellipse	Plein
Message	Triangle	Plein

(b) Symboles des relations SAMURAAIE

Relation	Profil	Style
Agrégation	Trait-point	Plein
Liaison	Flèche	Discontinu
Précédence	Flèche	Plein
Accès	Flèche	Plein
Copie	Flèche	Discontinu
Portée	Flèche	Pointillé
Fourniture	Flèche	Pointillé

TABLE 4.1 – Symboles des éléments et des relations SAMURAAIE

	Instances	Actions	Événements
Contenant	Ressources	Services	Capteurs
Contenu	Application	Programme	Messages
Association	Carte	Déploiement	Journal

TABLE 4.2 – Noms des abstractions

Abstraction	Agrégat	Information	Opération	Transmission
Ressources	Ordinateur	Mémoire	Processeur	Connecteur
Application	Processus	Blob	Activité	Flux

TABLE 4.3 – Noms des sommets d'instance

L'application comporte d'éventuels fichiers de données, des fichiers exécutables, et des processus, c'est-à-dire des blobs, des activités, et des flux. L'abstraction Application comporte donc des sommets d'agrégation Processus et des sommets simples Blob, Activité, et Flux. Le tableau 4.3 page 78 récapitule tous les noms des sommets du niveau Instances. Les topologies des ressources et de l'application sont abstraites par des arcs dont les types sont présentés ci-dessous.

Les caractéristiques des sommets sont les suivantes :

Agrégat Les sommets d'agrégation de contenant et de contenu, respectivement appelés Ordinateur et Processus, ont des caractéristiques. Les sommets Ordinateur ont un nom, les sommets Processus ont un identifiant et une ligne de commande.

Information Les sommets de contenant et de contenu d'information, respectivement appelés Mémoire et Blob, ont des caractéristiques de capacité ou taille, en octet, et de médium offert ou demandé.

Opération Les sommets de contenant et de contenu d'opération, respectivement appelés Processeur et Activité, ont des caractéristiques de fréquence offerte ou demandée, en Hertz, de jeu d'instructions offert ou demandé, et de capacité ou taille d'adressage, en bits. En l'occurrence, un processeur multi-cœur est abstrait par plusieurs sommets Processeur.

Transmission Les sommets de contenant et de contenu de transmission, respectivement appelés Connecteur et Flux, ont des caractéristiques de latence offerte ou demandée, en seconde, de débit offert ou demandé, en octet par seconde, de protocole offert ou demandé, et d'adresse ou chemin offert ou demandé.

Les sommets des abstractions du niveau Instances ont également une caractéristique commune : le nom de leur technologie. Cette caractéristique permet de différencier les instances en vue de leur gestion. Par ailleurs, les valeurs des caractéristiques de ces sommets peuvent rester non spécifiées.

Relations d'instance Les abstractions Ressources et Application introduisent deux nouveaux types de relations entre sommets : l'accès et la copie.

Les topologies de ressources et d'une application sont déterminées par les relations d'accès entre leurs éléments. Ces accès vont des processeurs aux connecteurs, des connecteurs aux mémoires, des activités aux flux, des flux aux blobs, et inversement. Afin de figurer ces relations, le niveau Instances introduit les arcs d'accès.

Les arcs de copie figurent les relations de copie entre empreintes en mémoire vive ou entre fichiers. Par exemple, si une application comporte un fichier binaire et qu'une copie de ce fichier est associée à plusieurs mémoires de masse, alors des arcs de copie connectent les sommets Blob du fichier original et des copies.

Services, Programme, et Déploiement

Le deuxième niveau d'abstraction, le niveau Actions, comporte les abstractions Services, Programme, et leur éventuelle association, Déploiement. De même qu'au niveau Instances, les sommets de ces abstractions spécialisent les sommets de base, et elles comportent de nouveaux types d'arcs.

Éléments d'action Les types de sommets des abstractions Services et Programme figurent le découpage en système d'exploitation et leur utilisation, introduit pour la classification présentée dans les deux chapitres précédents.

Les services comportent des systèmes d'exploitation, c'est-à-dire des services de mémorisation, d'exécution, et d'adressage. L'abstraction Services comporte donc des sommets d'agrégation Système et des sommets simples Service de mémorisation, d'exécution, et d'adressage.

Abstraction	Agrégat	Mémorisation	Exécution	Adressage
Services	Système	Service	Service	Service
Programme	Tâche	Requête	Requête	Requête

Table 4.4 – Noms des sommets d'action

Abstraction	Agrégat	Simple
Capteurs	Zone	Capteur
Messages	Modification	Message

Table 4.5 – Noms des sommets d'événement

Le programme comporte des tâches, c'est-à-dire des requêtes de mémorisation, d'exécution, et d'adressage. L'abstraction Programme comporte donc des sommets d'agrégation Tâche et des sommets simples Requête de mémorisation, d'exécution, et d'adressage. Le tableau 4.4 page 80 récapitule tous les noms des sommets du niveau Actions.

Une action peut avoir l'un des types suivants :

`CRÉER` Créer un contenu dans un contenant ;

`COPIER` Copier un contenu depuis un contenant vers un autre ;

`TESTER` Tester la présence d'un contenu dans un contenant ;

`SUSPENDRE` Suspendre le fonctionnement ou la modification d'un contenu dans un contenant ;

`REPRENDRE` Reprendre le fonctionnement ou la modification d'un contenu dans un contenant ;

`ENLEVER` Enlever un contenu d'un contenant.

De même qu'au niveau Instances, les sommets des abstractions du niveau Actions ont une caractéristique de technologie.

Relations d'action Les abstractions Services et Programme introduisent deux nouveaux types de relations entre sommets : la portée et la fourniture.

Les topologies de services et d'un programme sont déterminées par les relations de portée et de fourniture entre leurs éléments et ceux des abstractions du niveau Instances. Ces portées vont des systèmes d'exploitation aux ordinateurs, des services de mémorisation aux mémoires, des services d'exécution aux processeurs, des services d'adressage aux connecteurs, des tâches aux processus, des requêtes de mémorisation aux blobs, des requêtes d'exécution aux activités, et des requêtes d'adressage aux flux. Ces fournitures vont des processus aux systèmes d'exploitation et aux services de mémorisation, d'exécution, et d'adressage. Afin de figurer ces relations, le niveau Actions introduit donc les arcs de portée et de fourniture.

Capteurs, Messages, et Journal

Le troisième et dernier niveau d'abstraction est le niveau Événements. Il comporte trois abstractions appelées Capteurs, Messages, et leur éventuelle association, Journal. De même qu'au niveau Actions, les sommets de ces abstractions spécialisent les sommets de base. En revanche, elles ne comportent pas de nouveau type d'arcs par rapport à ceux déjà introduits au niveau Actions.

Les types de sommets des abstractions Capteurs et Messages découpent les changements des abstractions des niveaux Instances et Actions en zones de changement et modifications de ces zones.

Les capteurs se partagent les changements des abstractions des niveaux Instances et Actions. Ils délimitent ainsi des zones de changement. L'abstraction Capteurs comporte donc des sommets

d'agrégation Zone et des sommets simples Capteur. Les messages provoquent les changements des abstractions des niveaux Instances et Actions. Ils délimitent ainsi des modifications. L'abstraction Messages comporte donc des sommets d'agrégation Modification et des sommets simples Message. Le tableau 4.5 page 80 récapitule tous les noms des sommets du niveau Événements.

Un événement peut avoir l'un des types suivants :

`AJOUTER` Ajouter un élément ou une relation à une abstraction ;

`ÉDITER` Éditer un élément d'une abstraction ;

`ENLEVER` Enlever un élément ou une relation d'une abstraction.

De même qu'au niveau Actions, les sommets du niveau Événements ont des relations avec les sommets du niveau Instances mais également avec ceux du niveau Actions : des relations de portée. En effet, les événements portent sur des éléments de ressources, d'application, de systèmes d'exploitation, ou leur utilisation. Des arcs de portée connectent donc des sommets Capteur et Message à des sommets d'instance et d'action. Les événements pouvant concerner autant des sommets que des arcs, les messages peuvent comporter de l'information afin de spécifier le type et l'orientation d'un arc à ajouter.

4.3.3 Exemples

Cette sous-section illustre le modèle d'abstraction SAMURAAIE par trois exemples simples et complémentaires aux niveaux Instances, Actions, et Événements. Les symboles des éléments et des relations permettant de figurer les abstractions pour ces exemples sont décrits dans les tableaux 4.0(a) et 4.0(b) page 78.

Exécution d'un processus

Un exemple au niveau Instances est un ordinateur qui exécute un processus. La figure 4.3 page 82 illustre cet exemple. L'ordinateur comporte une mémoire vive, une mémoire de masse, un processeur, et des connecteurs intermédiaires. Le processus comporte une empreinte en mémoire vive, un fichier exécutable, une activité, et des flux intermédiaires. Les abstractions SAMURAAIE qui correspondent à un tel exemple sont les abstractions Ressources, Application, et Carte qui suivent (voir la figure 4.4 page 83).

Abstraction Ressources Cette abstraction comporte un sommet d'agrégation de type Ordinateur, sept sommets simples de types Mémoire, Processeur, et Connecteur, sept arcs d'agrégation, et huit arcs d'accès. Les caractéristiques d'état de tous les sommets valent `OK`. Les caractéristiques de chemin des connecteurs entre le processeur et la mémoire de masse valent `/tmp`.

Abstraction Application Cette abstraction comporte un sommet d'agrégation de type Processus, six sommets simples de types Blob, Activité, et Flux, sept arcs d'agrégation, et six arcs d'accès. Le fichier exécutable n'étant accédé qu'en lecture, cette abstraction comporte un arc d'accès de moins que celle de l'ordinateur. Les caractéristiques d'état de tous les sommets valent `OK` à l'exception de celui du fichier qui est `TERMINÉ`. Les caractéristiques de chemin des flux entre l'activité et le blob associé avec la mémoire de masse valent `TEST`.

Abstraction Carte Cette abstraction comporte tous les sommets des abstractions Ressources et Application ainsi que huit arcs de liaison. Ces arcs associent les éléments de contenant aux éléments de contenu correspondants.

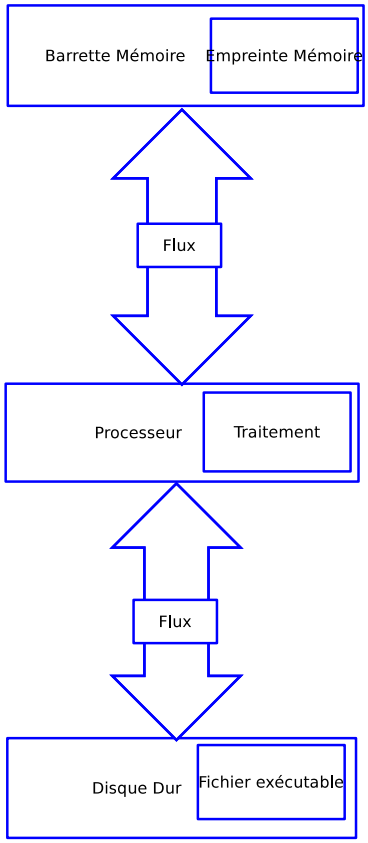


FIGURE 4.3 – Un ordinateur exécute un processus

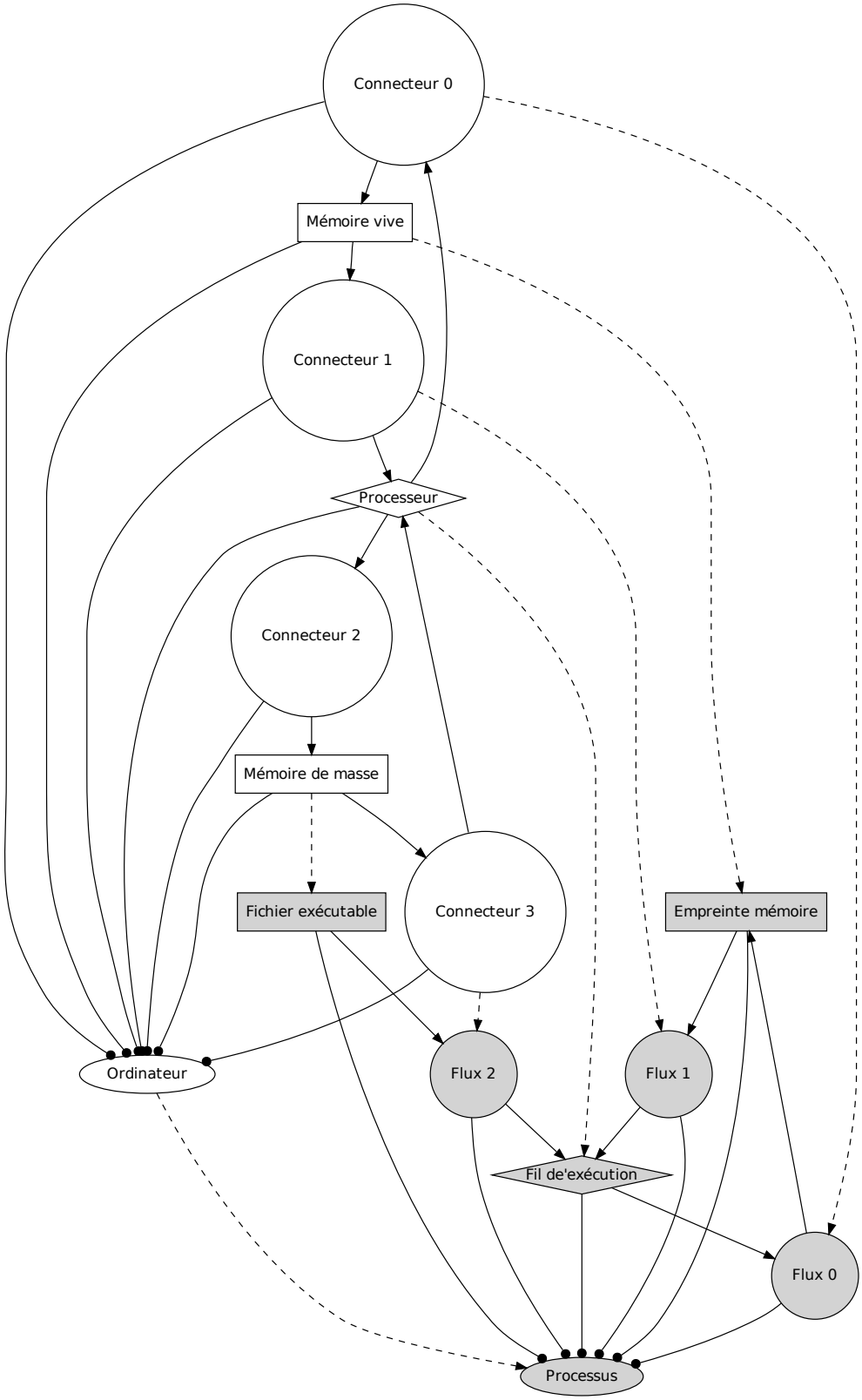


FIGURE 4.4 – Abstractions SAMURAAIE d'un ordinateur qui exécute un processus

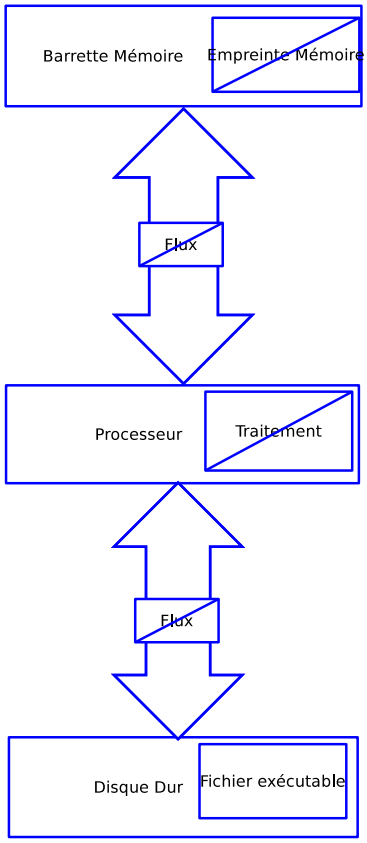


FIGURE 4.5 – Un arrêt de processus exécuté sur un ordinateur

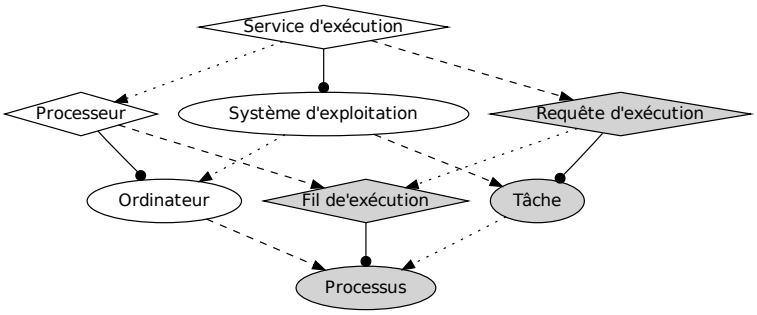


FIGURE 4.6 – Abstractions SAMURAAIE d'un arrêt d'un processus exécuté sur un ordinateur

Arrêt d'un processus en cours d'exécution

Un exemple au niveau Actions est l'arrêt du processus exécuté sur l'ordinateur de l'exemple précédent. La figure 4.5 page 84 illustre cet exemple. L'ordinateur est exploité par un système qui comporte un service d'exécution portant sur le processeur de cet ordinateur. La tâche comporte une requête d'exécution portant sur l'activité de ce processus. Les abstractions SAMURAAIE qui correspondent à cet exemple sont les abstractions Services, Programme, et Déploiement qui suivent (voir la figure 4.6 page 85).

Abstraction Services Cette abstraction comporte un sommet d'agrégation de type Système, un sommet simple de type Service d'exécution, les sommets de types Ordinateur et Processeur de l'abstraction Ressources, un arc d'agrégation, et deux arcs de portée. Les caractéristiques d'état de tous les sommets valent `OK`. La caractéristique de type d'action du service d'exécution vaut `ENLEVER`.

Abstraction Programme Cette abstraction comporte un sommet d'agrégation de type Tâche, un sommet simple de type Requête d'exécution, les sommets Processus et Activité de l'abstraction Application, un arc d'agrégation, et deux arcs de portée. Les caractéristiques d'état de tous les sommets valent `OK`. La caractéristique de type d'action de la requête d'exécution vaut `ENLEVER`.

Abstraction Déploiement Cette abstraction comporte tous les sommets des abstractions Services et Programme ainsi que deux arcs de liaison. Ces arcs associent les éléments de contenant aux éléments de contenu correspondants.

Ajout d'un processeur

Un exemple au niveau Événements est l'ajout d'un processeur à l'ordinateur des exemples précédents. La figure 4.7 page 86 illustre cet exemple. L'ordinateur est dans une zone qui comporte un capteur portant sur les connecteurs entre le processeur existant et les mémoires de cet ordinateur. La modification comporte cinq messages portant sur les connecteurs entre le processeur existent et les mémoires de cet ordinateur, d'une part, et sur le nouveau processeur, d'autre part. Les abstractions SAMURAAIE qui correspondent à cet exemple sont les abstractions Capteurs, Messages, et Journal qui suivent (voir la figure 4.8 page 87).

Abstraction Capteurs Cette abstraction comporte un sommet d'agrégation de type Zone, un sommet simple de type Capteur, les sommets de types Ordinateur, Processeur, et Connecteur de l'abstraction Ressources, un arc d'agrégation, et quatre arcs de portée. Les carac-

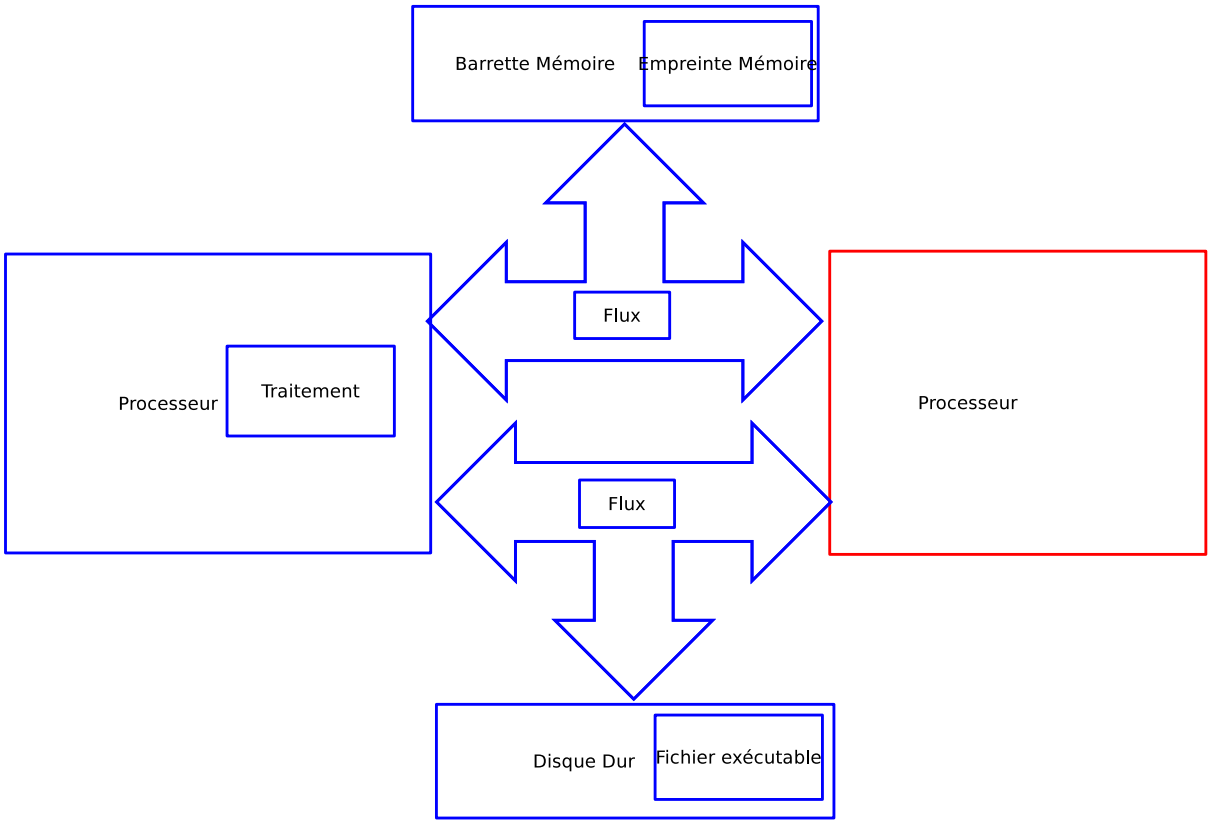


FIGURE 4.7 – Un ajout d'un processeur à un ordinateur

téristiques d'état de tous les sommets valent `OK`. La caractéristique de type d'événement du capteur vaut `AJOUTER`.

Abstraction Messages Cette abstraction comporte un sommet d'agrégation de type Modification, cinq sommets simples de type Message, les sommets Connecteur et le nouveau sommet Processeur de l'abstraction Ressources, un arc d'agrégation, neuf arcs de portée, et quatre arcs de précédence. Les caractéristiques d'état de tous les sommets valent `OK` à l'exception de celle du nouveau qui vaut `ENREGISTRÉ+` et des quatre messages précédés. Les caractéristiques de type d'événement des messages valent `AJOUTER` et l'information de type et d'orientation des arcs d'accès depuis et vers le nouveau processeur.

Abstraction Journal Cette abstraction comporte tous les sommets des abstractions Capteurs et Messages ainsi que six arcs de liaison. Ces arcs associent les éléments de contenant aux éléments de contenu correspondants.

4.3.4 Discussion

SAMURAAIE est un modèle qui permet d'abstraire l'ensemble des éléments informatiques d'un environnement de déploiement. Les ressources, l'application, les systèmes d'exploitation, l'utilisation de ces systèmes, et les changements de ces éléments peuvent effectivement être abstraits avec SAMURAAIE. Pour ce faire, ce modèle est fondé sur les trois distinctions suivantes :

Information, opération, et transmission Cette distinction reprend la structure générale de l'informatique illustrée par la figure 2.1 page 23.

Instances, actions, et événements Cette distinction permet d'abstraire autant les éléments informatiques que les actions qu'ils permettent et les événements qui les concernent.

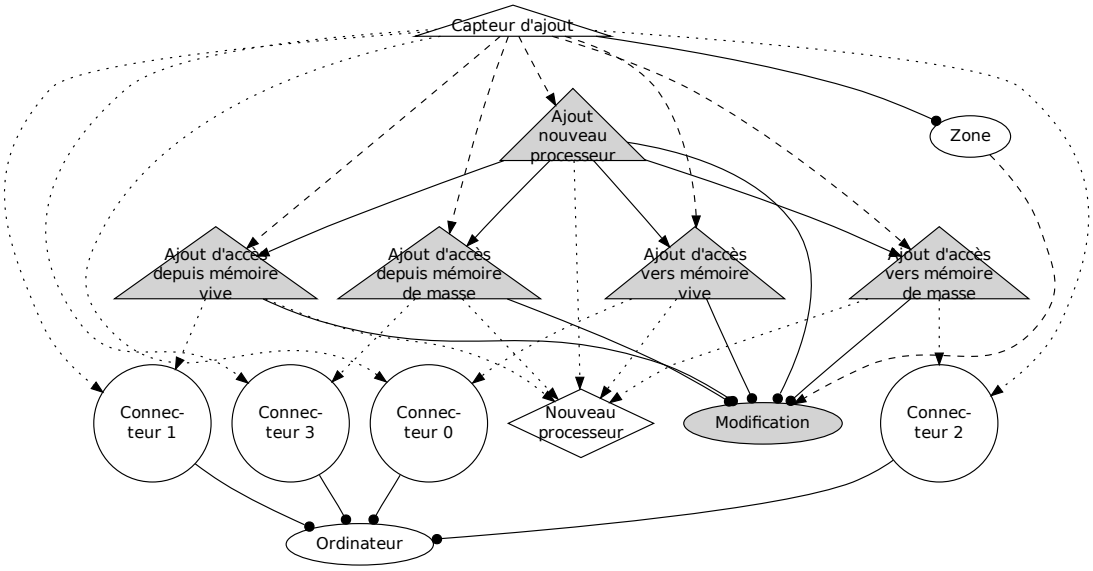


FIGURE 4.8 – Abstraction SAMURAAIE d'un ajout d'un processeur dans un ordinateur

Contenants et contenus Cette distinction permet d'exprimer des contraintes entre des réseaux d'éléments informatiques.

La séparation entre les éléments informatiques de transmission, d'une part, et les relations d'accès entre éléments, d'autre part, permet une abstraction fine des réseaux dans l'environnement de déploiement. En effet, il est ainsi possible d'abstraire le partage par plusieurs processeurs d'une même mémoire. Si ces processeurs utilisent un même connecteur, alors l'abstraction de la concentration de leurs accès comporte des arcs d'accès connectés à une seule paire de sommets Connecteur (un en lecture et l'autre en écriture), chacun de ces derniers ayant un débit maximum. Sinon, si ces processeurs utilisent des connecteurs différents, alors cette abstraction comporte plusieurs paires de sommets Connecteur. Il est donc également possible de différencier les débits, les latences, les protocoles, et les adresses ou chemins d'une même mémoire partagée par plusieurs processeurs.

Parmi les éléments informatiques de l'environnement de déploiement, qu'il est possible d'abstraire avec SAMURAAIE, figure l'implantation de SAMURAAIE elle-même. Destinée à l'utilisateur, cette implantation serait une application qui pourrait en effet nécessiter un déploiement sur des ressources distantes. Une partie de cette implantation devrait cependant fonctionner afin de prendre en charge cet auto-déploiement.

SAMURAAIE permet d'abstraire autant les contraintes architecturales que logistiques. Si la taille d'un fichier dépasse la capacité d'une mémoire par exemple, alors ce fichier et cette mémoire ne peuvent être associés. En revanche, si un fichier et une mémoire existants sont associés, qu'une copie attendue de ce fichier et une autre mémoire existante sont associés, et qu'un service de copie existe et a ces deux mémoires à sa portée, alors peuvent être associés ce service et une requête attendue de copie. Par ailleurs, les dépendances entre éléments d'une application, c'est-à-dire les relations de précédence, de copie, et d'accès déterminent les relations de précédence entre les éléments de programme portant sur eux.

SAMURAAIE permet également d'abstraire les changements d'un système informatique. D'une part, ce modèle offre un niveau Événements, afin d'abstraire les modifications des abstractions aux niveaux Instances et Actions. Ces dernières peuvent être découpées en différentes

zones, afin de pouvoir traiter des modifications concurrentes. D'autre part, la combinaison d'états et du temps dans les abstractions permet de disposer d'un horizon temporel. La durée de l'exécution voire de l'installation d'une application sur des ressources peut ainsi être estimée. La durée d'une action et de disponibilité d'une ressource pouvant être estimée ou connue, il est également possible d'éviter l'allocation de ressources dont la disponibilité ne suffirait pas pour la séquence installation et exécution.

Si SAMURAAIE permet d'abstraire les changements de l'environnement de déploiement, il ne permet cependant pas d'abstraire des éléments informatiques adaptables. Par exemple, le fait qu'une application change selon les ressources sur lesquelles elle est exécutée ne peut être abstrait. Réciproquement, le fait que des ressources changent selon l'application exécutée ne peut être abstrait. Ces cas sont cependant pris en compte dans l'architecture que propose ce chapitre, ils le sont toutefois grâce à la seconde partie de cette architecture : le modèle de déploiement sur demande.

SAMURAAIE est extensible de plusieurs manières. La première est d'enrichir ou de redéfinir la liste de valeurs d'une caractéristique qui en accepte. Par exemple, ajouter « amovible » à la liste de média des mémoires et des données qu'elles mémorisent. Une deuxième manière est d'ajouter une caractéristique. Par exemple, ajouter la version, la géolocalité, la consommation énergétique, le niveau de disponibilité, ou le niveau de sécurité. Une troisième manière est d'ajouter un nouveau type de relation entre éléments. La quatrième manière est de redéfinir les types des caractéristiques ou des arcs. Par exemple, rendre paramétrables les valeurs des caractéristiques.

SAMURAAIE est conçu pour abstraire des environnements de déploiement ordinaires dans le domaine de l'informatique à haute performance. Cependant, il permet d'abstraire des machines virtuelles, des processeurs graphiques, ou des caches de données par exemple. Il permet également d'abstraire des services de migration et la redondance de processus respectivement grâce aux actions et aux relations de copie. Par ailleurs, SAMURAAIE pourrait abstraire des applications de visualisation en abstrayant l'utilisateur comme un ordinateur, l'interface humain-machine étant un ensemble de connecteurs et l'information qu'elle transmet un ensemble de flux.

La quantité d'information à propos d'un environnement de déploiement que SAMURAAIE permet de collecter est grande. Cette collecte peut être effectuée par les administrateurs d'infrastructure ou par les développeurs d'application, voire effectuée automatiquement. Cette information peut en effet exister dans l'environnement sous d'autres formes, telles que par exemple dans un système d'information à propos de l'infrastructure informatique, ou dans les meta-données de l'application. Cette information peut également être obtenue, activement, par des techniques de sondage de l'infrastructure ou de profilage de l'application. À la limite, un peu d'information peut également être fournie par l'utilisateur lui-même.

Un objectif de la seconde partie de l'architecture que propose ce chapitre est justement d'entretenir les abstractions afin de gérer automatiquement le déploiement.

4.4 Modèle ODD d'automatisation du déploiement d'applications

ODD est un modèle d'automatisation du déploiement d'applications. Constitué d'acteurs, il offre une automatisation du déploiement pour des technologies à la fois parallèles, hétérogènes, et dynamiques. Pour ce faire, il utilise un double mécanisme de traduction et de connexion, de et avec son environnement. Ce mécanisme est fondé sur le modèle SAMURAAIE d'abstraction des systèmes informatiques, présenté dans la section précédente. Ces caractéristiques justifient le nom : *On-Demand Deployment* (ODD).

Cette section présente les quatre classes d'acteurs ODD que sont les gestionnaires de déploiement, les abstracteurs de déploiement, les traducteurs de technologie, et les connecteurs

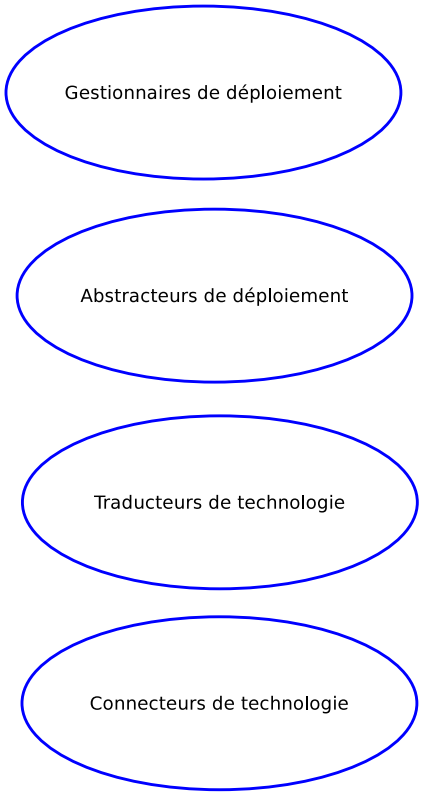


FIGURE 4.9 – Classes d'acteurs ODD

de technologie. Elle détaille ensuite les interactions des acteurs lors des différentes étapes du cycle de vie du modèle ODD. Enfin, elle discute ce modèle d'automatisation du déploiement d'applications.

4.4.1 Acteurs

Afin de gérer le déploiement sur demande, le modèle ODD articule des acteurs, lesquels se répartissent en quatre classes, voir la figure 4.9 page 89. Les gestionnaires ODD gèrent l'environnement de déploiement. Les abstracteurs ODD gèrent les abstractions SAMURAAIE de l'environnement de déploiement. Les traducteurs ODD traduisent l'information de technologie, non SAMURAAIE, en abstraction SAMURAAIE, et réciproquement. Enfin, les connecteurs ODD assurent la communication entre les traducteurs ODD et les technologies de l'environnement de déploiement.

Gestionnaires

Dans le modèle ODD, l'automatisation du déploiement d'applications revient principalement à quatre acteurs, voir la figure 4.10 page 90. Le premier associe des contenants et des contenus d'instance, d'action, et d'événement, il est appelé Associateur. Le deuxième conseille des actions à partir d'associations de contenants et de contenus d'instance, il est appelé Conseiller. Le troisième exécute des associations de contenants et de contenus d'action, il est appelé Exécuteur. Enfin, le quatrième et dernier traite les associations de contenants et de contenus d'événement, il est appelé Contrôleur.

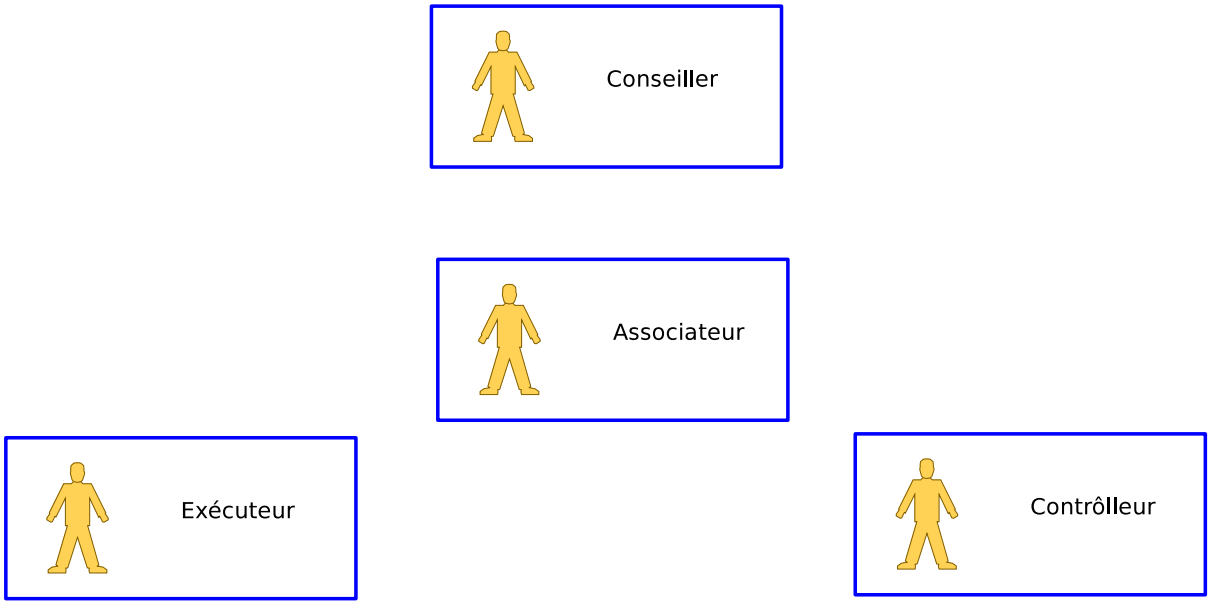


FIGURE 4.10 – Gestionnaires ODD de déploiement

Associateur Le gestionnaire Associateur décide si des éléments de contenant conviennent aux éléments de contenu, d'une part, et gère les associations de ces éléments, d'autre part. Il prend en charge l'activité de l'utilisateur consistant à choisir des ressources pour l'application, des services pour les requêtes, et des capteurs pour les messages. Afin de prendre sa décision, il consulte autant les valeurs des caractéristiques des éléments considérés que leurs relations, autrement dit, les sommets et les arcs qui leur sont incidents importent autant. Les éléments de contenant et de contenu doivent d'abord être au même niveau d'abstraction. Leurs états de fonctionnement doivent être respectivement opérationnel (`OK`) et en attente (`ATTENDU`). L'élément de contenu ne doit pas encore être associé. Associateur vérifie également que ces éléments existent dans la même fenêtre temporelle.

Des éléments de contenant et de contenu peuvent réussir les tests préalables indiqués ci-dessus sans pour autant que le gestionnaire Associateur puisse les associer. Pour cela, des tests correspondants à leur niveau d'abstraction doivent également réussir. Au niveau Instances, les caractéristiques offertes par les éléments de contenant doivent satisfaire les demandes des éléments de contenu. Des éléments de contenant qui ne seraient pas connectés par des arcs d'accès ne peuvent pas être associés à des éléments de contenu qui le seraient. Aux niveaux Actions et Événements, ce sont les types offerts et demandés, d'une part, et les relations de portée, d'autre part, qui doivent correspondre.

En associant des éléments de contenant aux éléments de contenu qui le demandent, le gestionnaire Associateur met en œuvre une stratégie. Cette stratégie peut par exemple être de minimiser la durée d'exécution de l'application, de minimiser la durée du processus de déploiement pris dans son ensemble, ou de minimiser les transferts de données. Cette stratégie peut par exemple impliquer que Associateur évite d'associer un même élément de contenant à plusieurs éléments de contenu ou qu'il ajoute des copies de données ou d'activités dans l'abstraction de l'application.

Dans certaines conditions, aucune association n'est possible ou, du moins, les associations possibles sont insuffisantes. Cela advient quand les ressources sont incompatibles avec l'application, quand les services sont incapables de réaliser le programme d'installation et d'exécution de l'application, ou quand aucun capteur ne peut traiter les modifications demandées. Dans ces

conditions, le gestionnaire Associateur attend que d'autres acteurs modifient les abstractions jusqu'à ce que ces conditions changent.

En gérant les associations entre éléments de contenant et de contenu à tous les niveaux d'abstraction, le gestionnaire Associateur consulte toutes les abstractions SAMURAAIE et modifie les abstractions Carte, Déploiement, et Journal, voire Application quand il ajoute des copies de données ou d'activité. Cela lui confère un rôle central parmi les quatre gestionnaires.

Conseiller Le gestionnaire Conseiller décide des actions à entreprendre. Il prend en charge l'activité de l'utilisateur consistant à transformer en requêtes les choix de ressources pour l'application. Afin de prendre sa décision, il consulte les états de fonctionnement des éléments associés au niveau Instances, d'une part, et les relations de ces éléments, d'autre part. Il n'entreprend aucune action pour des éléments sur lesquels porteraient déjà des actions en attente ou opérationnelles. Une fois sa décision prise, il ajoute des requêtes de service dans l'abstraction Programme et attend que les autres acteurs modifient les abstractions jusqu'à ce que les associations d'éléments au niveau Instances changent.

Les actions que le gestionnaire Conseiller entreprend ont un objectif : déployer, contrôler, ou reployer. Si les éléments de contenu associés sont en attente, alors l'objectif des actions est de les déployer. Par exemple, l'objectif d'une action portant sur une activité en attente est de la créer, d'une copie en attente d'un fichier est de copier ce fichier. S'ils sont opérationnels, alors l'objectif est de les contrôler. Par exemple, l'objectif d'une action portant sur un flux opérationnel est de tester son existence. S'ils sont terminés, arrêtés, proscrits, ou en erreur, alors l'objectif est de les reployer. Par exemple, l'objectif d'une action portant sur un processus opérationnel associé à un ordinateur proscrit est d'arrêter ce processus.

Le gestionnaire Conseiller détermine également les relations de précédence entre les actions entreprises. Il le déduit des abstractions au niveau Instances elles-mêmes. Par exemple, une activité ne peut être crée sur un processeur qu'après que tous les fichiers auxquels elle accède soient opérationnels, d'une part, et associés avec des mémoires auxquelles ce processeur a accès, d'autre part. Un fichier ne peut être copié qu'après que les activités qui l'accèdent en écriture ne soient plus opérationnelles. Conseiller connecte donc les requêtes de service qu'il ajoute avec des arcs de précédence.

Le gestionnaire Conseiller peut résoudre des problèmes d'état de fonctionnement. Un tel problème advient, par exemple, quand un fichier est en attente mais que le processus qui l'écrit n'est pas lui-même en attente ou bien a échoué. De même, afin d'écrire ce fichier, ce processus a besoin de lire d'autres fichiers et ceux-ci peuvent également ne pas être dans un état opérationnel. Si remonter ce flot ne convergeait pas vers des éléments opérationnelles, alors il n'y aurait aucun moyen d'écrire ce fichier. Sinon, Conseiller peut changer l'état de ces éléments d'application afin qu'ils passent ou repassent en attente.

Le gestionnaire Conseiller peut également résoudre des problèmes de portée. Un tel problème advient, par exemple, quand aucun service de copie n'a à sa portée les élément de contenant associés respectivement à un élément de contenu et à une de ses copies. Conseiller peut alors chercher des éléments de contenant intermédiaires, à la portée de services de copie ayant deux à deux un élément de contenant commun à leur portée. S'il en trouvait, alors il entreprendrait autant de copies intermédiaires que nécessaires, c'est-à-dire qu'il déconnecterait la copie déjà associée de l'original, connecterait de nouvelles copies jusqu'à celle-ci, et ajouterait les requêtes de copie qui iraient alors bien.

En entreprenant des actions, le gestionnaire Conseiller consulte les abstractions des niveaux Instances et Actions. Il ne modifie que l'abstraction Programme. En transformant les associations au niveau Instances par des éléments de contenu au niveau Actions, Conseiller collabore avec Associateur.

Exécuteur Le gestionnaire Exécuteur réalise les actions. Il prend en charge l'activité de l'utilisateur consistant à transformer en actes les choix de services pour les requêtes. Pour ce faire, il consulte les états de fonctionnement des éléments associés au niveau Actions, d'une part, et les relations de précédence de ces éléments, d'autre part. Il ne réalise aucune action qui ne serait pas en attente. Une fois connues, il rend opérationnelles les requêtes dans l'abstraction Programme, et attend que les autres acteurs modifient les abstractions jusqu'à ce que les associations d'éléments au niveau Actions changent.

Le gestionnaire Exécuteur organise la réalisation du programme d'installation et d'exécution de l'application. À la manière d'un moteur d'exécution de flots de travail, il fonctionne en deux étapes : analyser les relations de précédence entre les actions à réaliser, et réaliser parallèlement celles qui ne sont pas précédées. Il boucle sur ce fonctionnement jusqu'à ce qu'il n'y ait plus aucune action à réaliser dans le programme.

En réalisant des actions, le gestionnaire Exécuteur consulte les trois abstractions du niveau Actions et ne modifie que l'abstraction Programme. En activant les requêtes d'abord entreprises puis associées, Exécuteur collabore avec Conseiller et avec Associateur.

Contrôleur Le gestionnaire Contrôleur traite les événements. Il prend en charge l'activité de l'utilisateur consistant à traiter les messages. Pour ce faire, il consulte les états de fonctionnement des éléments associés au niveau Événements, d'une part, et les relations de précédence de ces éléments, d'autre part. Il ne traite aucun événement qui ne serait pas en attente. Une fois connus, il rend opérationnels les messages, les traite, les déclare terminés, et attend que les autres acteurs modifient les abstractions jusqu'à ce que les associations d'éléments au niveau Événements changent.

De même que Exécuteur, le gestionnaire Contrôleur organise le traitement des événements à la manière d'un moteur d'exécution de flots de travail. Il traite tous les événements, c'est-à-dire que, sans lui, les abstractions aux niveaux Instances et Actions ne changeraient pas. En effet, les gestionnaires Associateur, Conseiller, et Exécuteur ne modifient pas directement les abstractions mais ajoutent des événements pour cela. Contrôleur traite alors ces événements en attente.

Pour chaque abstraction, le gestionnaire Contrôleur définit le nombre et la portée de capteurs. Afin que des événements puissent advenir aux frontières de ces portées, il s'assure que ces portées se recouvrent. Inversement, afin qu'aucun événement n'advienne sur une partie d'une abstraction, il s'assure que cette partie soit hors de portée des capteurs. Enfin, afin de mieux répartir la charge des événements entre les capteurs, il peut en changer le nombre et la portée.

En traitant des événements, le gestionnaire Contrôleur consulte et modifie toutes les abstractions.

Abstracteurs

Les abstracteurs maintiennent les abstractions SAMURAAIE. Ils prennent en charge l'activité de l'utilisateur consistant à mémoriser et à restituer l'information acquise à propos de l'environnement. Il existe autant d'abstracteurs que d'abstractions : abstracteurs Ressources, Application, Carte, Services, Programme, Déploiement, Capteurs, Messages, et Journal. Quelconque acteur voulant consulter ou modifier une abstraction doit interagir avec un abstracteur. Cependant, étant elle-même abstraite au niveau Événements, la modification des abstractions aux niveaux Instances et Actions est indirecte. À toute modification doit effectivement correspondre un ensemble de messages, que des gestionnaires associeront à des capteurs puis traiteront.

Par l'information qu'ils maintiennent, les abstracteurs sont intermédiateurs des gestionnaires et des traducteurs. Quand l'un de ces derniers modifie une abstraction, l'abstracteur accumule des messages dans une transaction, jusqu'à ce que l'acteur considéré consigne ou annule sa

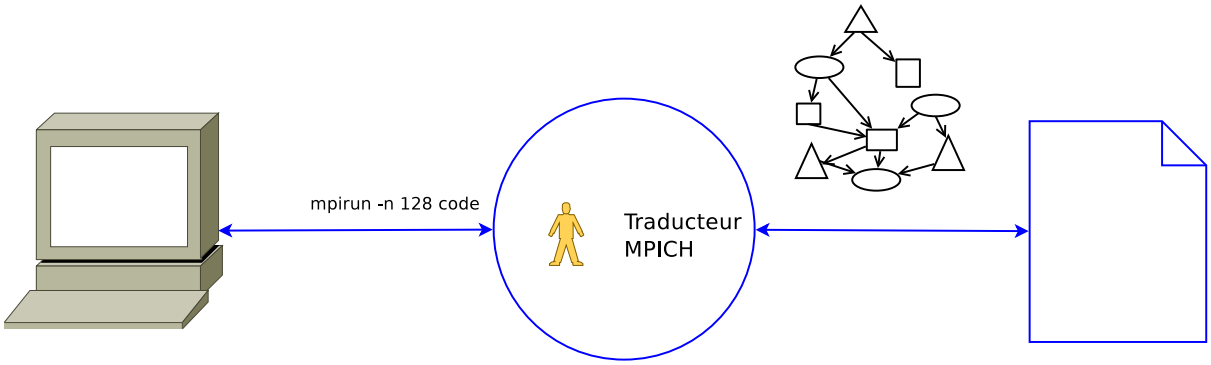


FIGURE 4.11 – Exemple du traducteur d'application MPICH

modification. S'il annule cette modification, l'abstracteur supprime les messages. S'il la consigne, l'abstracteur ajoute les messages dans l'abstraction Messages et deux cas peuvent alors advenir. Soit le gestionnaire Contrôleur les traite sans problème et alors l'abstracteur considéré avertit tous les acteurs ayant consulté ou modifié l'abstraction considérée, soit il défait cette modification et n'avertit que l'acteur considéré.

Traducteurs

Les traducteurs ODD traduisent de l'information en abstraction SAMURAAIE et réciproquement. Ils prennent en charge l'activité de l'utilisateur consistant à se représenter l'information acquise à propos de l'environnement et à expliciter ses décisions auprès de celui-ci. Ils modifient les abstractions afin de refléter l'information qu'ils reçoivent, d'une part, et émettent de l'information afin de refléter les modifications de ces abstractions, d'autre part. Au passage, ils peuvent mémoriser de l'information, ce qui peut permettre de limiter d'éventuelles ambiguïtés de traduction. Afin d'en limiter la complexité et d'en favoriser la réutilisation, un traducteur ne traduit généralement que pour un couple information-abstraction. Ce couple est donc caractéristique.

Les traducteurs d'application abstraient celle-ci complètement, c'est-à-dire à partir des éléments opérationnels a priori : des archives par exemple. En d'autres termes, ces traducteurs abstraient non seulement les éléments demandés de l'application, mais également les éléments dont elle dépend afin d'être exécutée. Pour reprendre cet exemple, si le fichier existant est une archive et que le fichier exécutable de l'application demandée s'y trouve, alors le traducteur doit abstraire le processus de désarchivage en abstrayant l'application.

Les traducteurs de déploiement traduisent les actions réalisées par le gestionnaire Exécuteur en information pour les connecteurs de programme et de services. De même que les autres, ces traducteurs font le relais entre la logique de déploiement et celles des technologies impliquées.

Par exemple, un traducteur d'application MPICH saurait construire l'abstraction SAMURAAIE d'une application MPICH à partir des paramètres d'une commande `mpirun` et saurait l'ajouter dans l'abstraction Application. Réciproquement, ce traducteur saurait construire les paramètres d'une commande `mpirun` à partir d'une abstraction SAMURAAIE d'une application MPICH ajoutée dans l'abstraction Application. La figure 4.11 page 93 illustre cet exemple.

Connecteurs

Les connecteurs ODD interfacent le modèle d'automatisation du déploiement d'applications et l'environnement de déploiement. Ils prennent en charge l'activité de l'utilisateur consistant à acquérir de l'information à propos de l'environnement et à agir sur celui-ci. Ils connectent des traducteurs, nécessairement implantés dans une technologie particulière, et les technologies

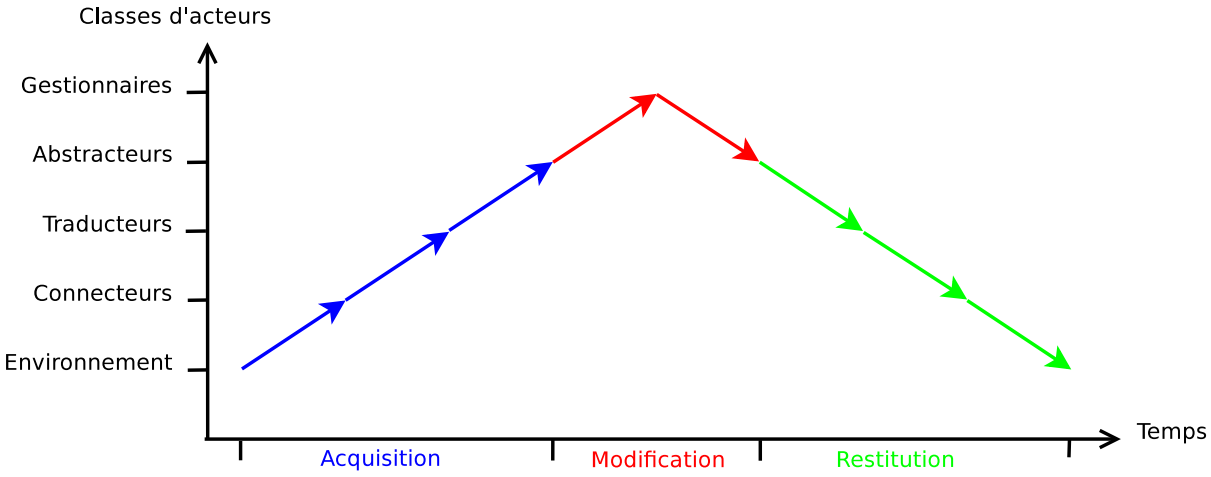


FIGURE 4.12 – Interactions des classes d'acteurs ODD

autant de l'infrastructure informatique que de l'application voire d'interface avec l'utilisateur. En d'autres termes, ils constituent une double interface de programmation. La première est destinée à l'environnement, afin que celui-ci manipule l'implantation du modèle ODD. La seconde est destinée à cette implantation, afin qu'elle manipule l'environnement. La technologie supportée caractérise un connecteur.

Les connecteurs ODD peuvent être actifs de plusieurs manières. D'abord, ils distribuent aux traducteurs l'information obtenue de l'environnement, et réciproquement. Les connecteurs de ressources ou de services peuvent par exemple indiquer aux traducteurs d'application que sont déjà mémorisés des fichiers exécutables élémentaires, tels que des outils d'archivage ou des compilateurs. Ils peuvent également utiliser les abstractions afin d'adapter une partie de l'environnement au reste de celui-ci. Par exemple, un connecteur de ressources peut en réserver à partir du nombre d'activités en attente. De même, un connecteur d'application peut choisir une implantation à partir de l'architecture de jeu d'instructions des processeurs opérationnels mais non associés.

En connectant l'implantation du modèle ODD d'automatisation du déploiement d'applications à l'environnement de déploiement, les connecteurs ODD consultent et modifient toutes les abstractions des niveaux Instances et Actions. Parmi tous les acteurs ODD, ils ne collaborent qu'avec les traducteurs.

4.4.2 Interactions

Parce qu'ils sont concurrents, à la manière du modèle d'acteurs [37], les acteurs ODD interagissent en communiquant par passage de messages. Cependant, ces messages d'interaction ne doivent pas être confondus avec les messages que permet d'abstraire le modèle SAMURAAIE. Afin d'affirmer la séparation de leurs préoccupations, aucun acteur ne communique avec des acteurs des quatre classes à la fois. Les gestionnaires communiquent avec les abstracteurs, les abstracteurs avec les gestionnaires et les traducteurs, les traducteurs avec les abstracteurs et les connecteurs, et les connecteurs avec les traducteurs et l'environnement.

Cette sous-section présente les interactions des classes d'acteurs, des acteurs, et d'acteurs pour la modification d'une abstraction. Elle présent également les séquences de démarrage et d'arrêt du modèle ODD.

Classes d'acteurs

En fonctionnement ordinaire, les interactions d'acteurs peuvent être réparties entre les trois étapes Acquisition, Modification, et Restitution du principe de permanence du modèle ODD présenté plus haut (voir la figure 4.1 page 74). Il est important de bien percevoir la concurrence possible entre ces interactions. Par exemple, des interactions d'acquisition et de restitution peuvent avoir lieu simultanément. Ces interactions de classes d'acteurs sont les suivantes (voir la figure 4.12 page 94) :

Acquisition L'environnement manipule les connecteurs ; les connecteurs obtiennent de l'information à propos de l'environnement, ils la distribuent aux traducteurs ; les traducteurs traduisent l'information et la transmettent aux abstracteurs.

Modification Les abstracteurs obtiennent de l'information à propos des abstractions, ils en avertissent les gestionnaires ; les gestionnaires obtiennent de l'information à propos des abstractions, ils produisent de l'information à propos des abstractions et la transmettent aux abstracteurs ;

Restitution Les abstracteurs obtiennent de l'information à propos des abstractions, ils en avertissent les traducteurs ; les traducteurs obtiennent de l'information à propos des abstractions, ils la traduisent et la transmettent aux connecteurs ; les connecteurs obtiennent de l'information à propos de l'environnement, ils manipulent l'environnement.

Acteurs

Chaque information acquise est d'abord traduite puis mémorisée dans une abstraction. Le mécanisme de prise de décision analyse alors cet événement et peut décider d'une action. Quand une action a été décidée, elle est d'abord traduite pour l'environnement puis réalisée. Dans le détail, les interactions des acteurs sont les suivantes (voir figure 4.13 page 96) :

- Les connecteurs appellent les traducteurs de ressources, de services, et d'application ;
- Les traducteurs de ressources, de services, et d'application appellent les abstracteurs afin de modifier leurs abstractions respectives ;
- Le gestionnaire Associateur est averti par les abstracteurs des modifications, appelle les abstracteurs de ressources et d'application afin de consulter leurs abstractions respectives, et appelle l'abstracteur de carte afin de modifier son abstraction ;
- Le gestionnaire Conseiller est averti par les abstracteurs des modifications, appelle les abstracteurs d'application, de carte, et de services afin de consulter leurs abstractions respectives, et appelle l'abstracteur de programme afin de modifier son abstraction ;
- Le gestionnaire Associateur est averti par les abstracteurs des modifications, appelle les abstracteurs de services et de programme afin de consulter leurs abstractions respectives, et appelle l'abstracteur de déploiement afin de modifier son abstraction ;
- Le gestionnaire Exécuteur est averti par les abstracteurs des modifications, appelle les abstracteurs de programme et de déploiement afin de consulter leurs abstractions respectives, et appelle l'abstracteur de déploiement afin de modifier son abstraction ;
- Les traducteurs de déploiement sont avertis par les abstracteurs des modifications, appellent l'abstracteur de déploiement afin de consulter son abstraction, et appellent les connecteurs afin de les avertir des modifications.

Modification d'abstraction

Afin d'assurer la cohérence des abstractions avec l'environnement, les événements sont abstraits avant d'être traités. Dans le détail, les interactions d'acteurs ODD pour la modification d'une abstraction sont les suivantes (voir la figure 4.14 page 97) :

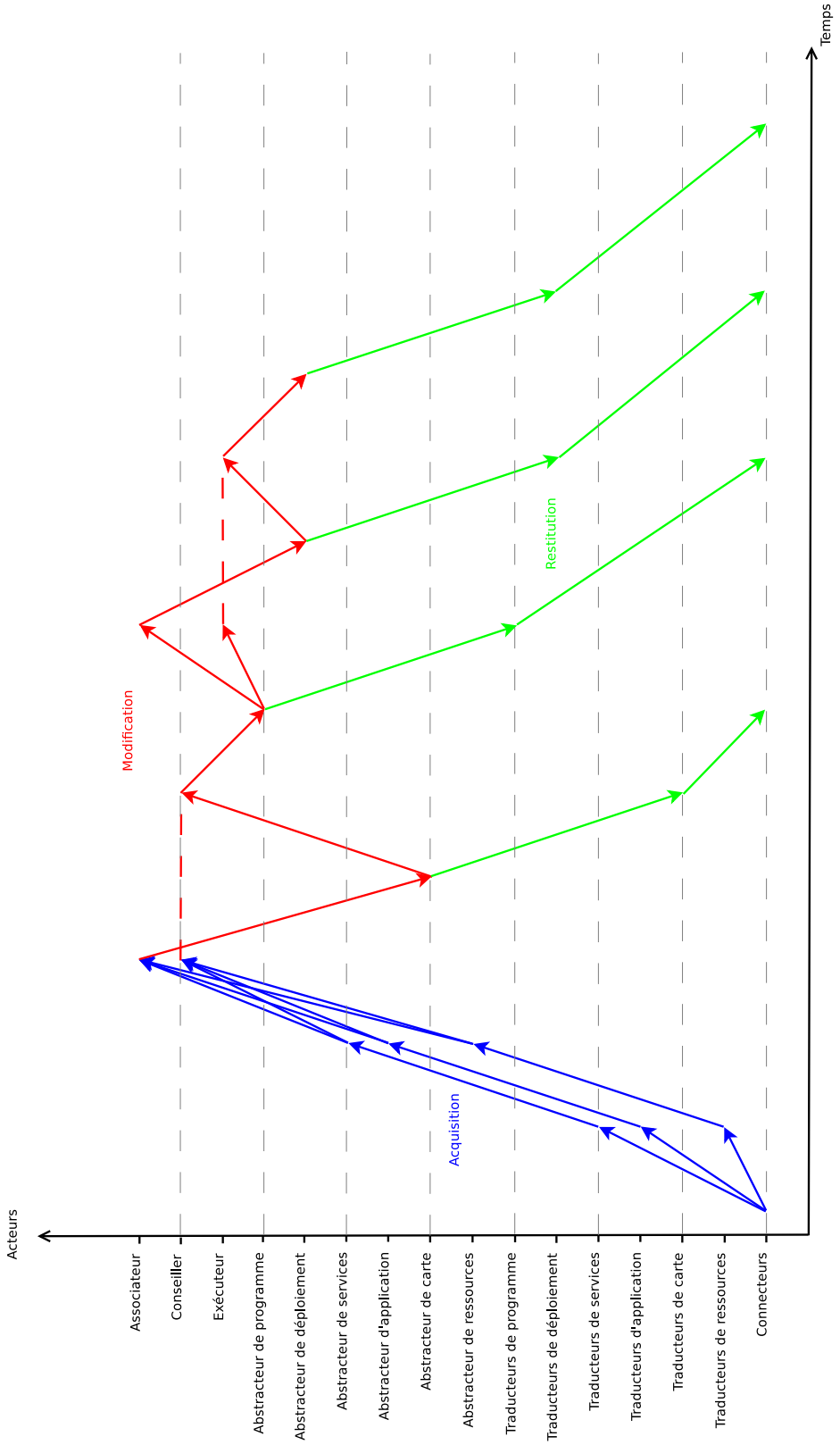


FIGURE 4.13 – Interactions simplifiées des acteurs ODD

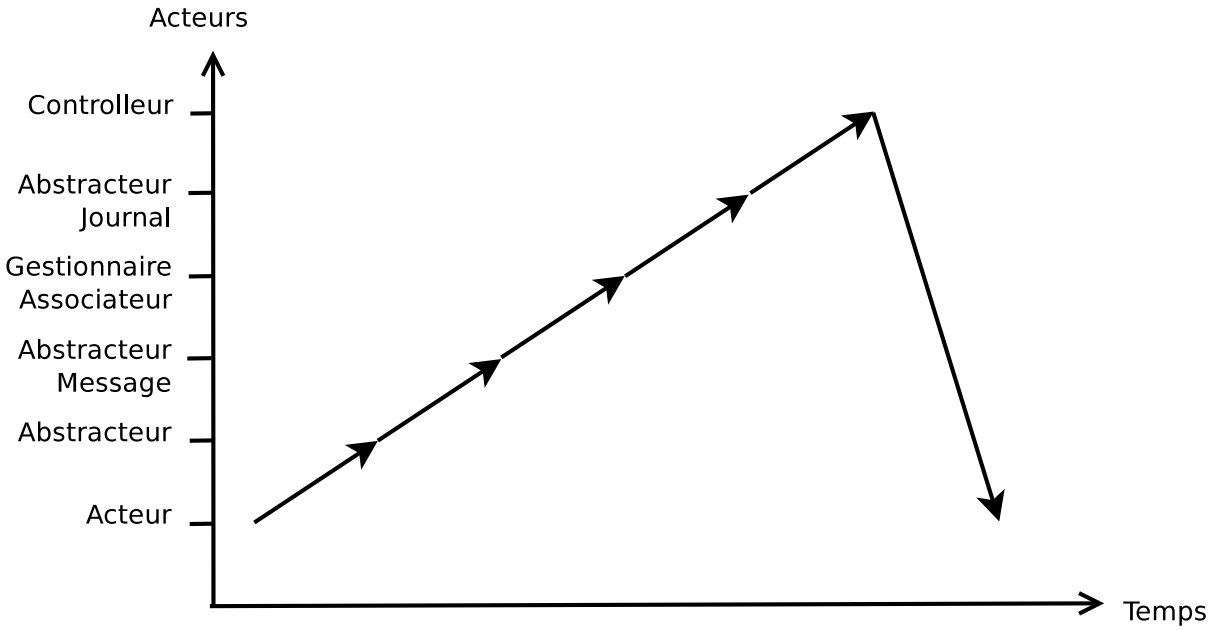


FIGURE 4.14 – Interactions d'acteurs ODD pour la modification d'une abstraction

- Un premier acteur appelle un premier abstracteur afin de modifier une première abstraction ;
- Le premier abstracteur appelle l'abstracteur de messages afin d'abstraire cette modification ; il ajoute une modification, des messages, et les relations d'agrégation entre ces messages et cette modification, de précédence entre ces messages, et de portée entre ces messages et des éléments de la première abstraction ;
- L'abstracteur de messages crée une nouvelle transaction et y accumule cette modification, ces messages, et ces relations ;
- Si le premier acteur annule sa modification de la première abstraction, alors le premier abstracteur annule à son tour sa modification de l'abstraction de messages ; sinon, lorsque le premier acteur a terminé, il consigne sa modification de la première abstraction et le premier abstracteur consigne à son tour sa modification de l'abstraction de messages ;
- L'abstracteur de messages vide la transaction dans son abstraction et avertit le gestionnaire Associateur ;
- Le gestionnaire Associateur appelle l'abstracteur de journal afin d'ajouter les relations de liaison entre des capteurs et les messages, et entre la zone et la modification auxquelles ces capteurs et messages appartiennent respectivement ;
- L'abstracteur de journal crée une nouvelle transaction et y accumule ces relations de liaison ;
- Lorsque le gestionnaire Associateur a associé tous les messages et la modification à laquelle ces messages appartiennent, il consigne sa modification de l'abstraction de journal ;
- L'abstracteur de journal vide la transaction dans son abstraction et avertit le gestionnaire Contrôleur ;
- Le gestionnaire Contrôleur connaît les messages et la modification à laquelle ces messages appartiennent, tous associés, d'une part, et les relations d'agrégation, de précédence, et de portée de ces messages, d'autre part ; en suivant les relations de précédence et de portée, il rend opérationnels et traite les messages portant sur la première abstraction, en modifiant directement celle-ci. S'il échoue en traitant un message, alors il le marque en échec, suit les relations d'agrégation et défait et marque suspendu tous les messages déjà traités et

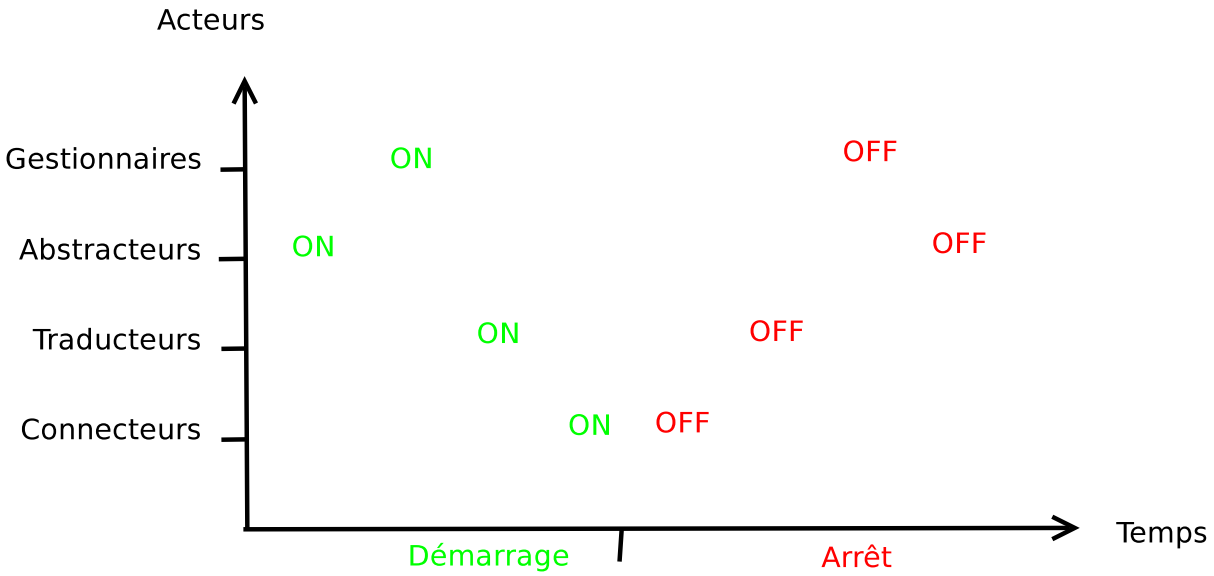


FIGURE 4.15 – Séquences de démarrage et d'arrêt du modèle ODD

	Déploiement	Technologie
Gestion	Gestionnaires	Connecteurs
Données	Abstracteurs	Traducteurs

TABLE 4.6 – Double séparation des préoccupations en classes d'acteurs ODD

agrégés dans la même modification que le message considéré ; sinon il marque ce message comme étant terminé ; quand il a traité tous les messages de la modification, il la marque terminée ;
- Le premier abstracteur, qui connaît son abstraction, constate que cette modification est terminée et en avertit le premier acteur.

Démarrage et arrêt du modèle

Finalement, bien que le modèle ODD ait vocation à fonctionner en permanence, une étape de démarrage est nécessaire et une autre d'arrêt peut le devenir. La séquence de démarrage est alors la suivante : dans l'ordre chronologique sont activés les abstracteurs, les gestionnaires, les traducteurs, et les connecteurs. Pour l'étape d'arrêt, la séquence est inversée : toujours dans l'ordre chronologique sont désactivés les connecteurs, les traducteurs, les gestionnaires, et les abstracteurs. La figure 4.15 page 98 illustre ces deux séquences. Si l'implantation de ce modèle est abstraite avec SAMURAAIE, alors ces étapes seraient elles-mêmes abstraites par des actions de création et d'arrêt portant sur une instance de cette implantation.

4.4.3 Discussion

Le modèle ODD propose de réduire la complexité des préoccupations de l'automatisation du déploiement en séparant doublement celles-ci en classes d'acteurs. Le premier axe de séparation distingue la logique de déploiement, représentée par les gestionnaires et les abstracteurs, et les logiques des technologies impliquées, représentées par les traducteurs et les connecteurs. Le second axe distingue la gestion, représentée par les gestionnaires et les connecteurs, et les données, représentées par les abstracteurs et les traducteurs. Le tableau 4.6 page 98 récapitule

cette double séparation.

La logique de déploiement comporte deux parties. Dans la première, les gestionnaires Associateur et Conseiller décident conjointement de l'ordonnancement, c'est-à-dire autant l'objectif à atteindre, au niveau Instances, que la manière de l'atteindre, au niveau Actions. Cette vision de l'ordonnancement contraste avec celle des systèmes d'exploitation, des gestionnaires de lots, ou des moteurs d'exécution de flots de travail parce qu'elle ne suppose pas que la manière d'atteindre un objectif et l'objectif lui-même soient indépendants. Dans la seconde partie, les gestionnaires Exécuteur et Contrôleur se répartissent la réalisation et le suivi de ces ordonnances.

De la planification à la rétroaction, les gestionnaires de déploiement peuvent mettre en œuvre des stratégies diverses. La stratégie de planification supposerait extrêmement que toutes les actions soient prévues et qu'aucun événement imprévu n'advienne. La stratégie de rétroaction supposerait inversement qu'aucune action ne soit prévue et que des événements adviennent. Dans la première, les gestionnaires Associateur et Conseiller fixeraient l'ordonnancement une fois pour toutes alors que, dans la seconde, ils le remettraient toujours en cause. Les gestionnaires ODD peuvent mettre en œuvre une de ces stratégies ou, plus probablement, une stratégie intermédiaire.

L'équilibre entre planification et rétroaction dans la stratégie des gestionnaires est important. En effet, la stratégie de planification peut permettre l'optimisation de chaque étape du déploiement. En revanche, elle peut ne jamais converger, à cause de l'indéterminisme du modèle d'acteurs ou de la complexité de la planification elle-même. De même, la stratégie de rétroaction peut permettre d'augmenter l'efficacité mais elle peut également ne jamais converger, à cause du grand nombre d'actions possibles. Dans le modèle, cet équilibre est déterminé par la quantité d'événements consignés entre chaque modification.

Les logiques des technologies étant séparées de la logique de déploiement, elles peuvent être multipliées à volonté. Par ailleurs, elles influencent la logique de déploiement en décidant de la quantité d'information qu'elles transmettent aux abstracteurs, d'une part. D'autre part, si une partie de la logique de déploiement se trouvait dans l'environnement, alors elles peuvent en abstraire les décisions. Par exemple, la décision d'un associateur externe, tel que l'utilisateur, peut être transmise par un connecteur qui la distribuerait à un traducteur de déploiement.

4.5 Conclusion

Les modèles complémentaires SAMURAAIE et ODD constituent une architecture de déploiement dynamique d'application, architecture capable de résoudre des situations complexes de déploiement, à la place de l'utilisateur. En proposant un modèle d'abstraction à trois niveaux (instances, actions, et événements) des systèmes informatiques, SAMURAAIE permet de considérer le déploiement non plus comme un processus linéaire mais comme un cycle permanent. Fondé sur SAMURAAIE, le modèle d'automatisation ODD propose de distinguer la logique de déploiement et les logiques des technologies impliquées, afin de fonctionner dans des environnements de déploiement parallèles, hétérogènes, et dynamiques, tels qu'en informatique à haute performance.

Chapitre 5

Prototype ANGE

5.1 Introduction

L'architecture ODD/SAMURAAIE, que propose le chapitre précédent, permet de déployer dynamiquement une application dans le contexte fortement contraint de l'informatique à haute performance. Elle comporte deux modèles complémentaires. Le premier utilise des graphes orientés afin d'abstraire le système. Le second définit des acteurs afin d'automatiser le déploiement. Cette architecture ne précise cependant pas comment elle pourrait être implantée.

Ce chapitre propose une implantation prouvant le concept d'architecture de déploiement dynamique d'application. Il commence par en donner les généralités. Puis il en explique successivement les deux parties, correspondant aux modèles complémentaires SAMURAAIE et ODD. Il explique ensuite comment le support de nouvelles technologies peut être ajouté dans cette implantation. Enfin, il discute cette implantation.

5.2 Généralités

Appelé « Adage Nouvelle Génération » (*Adage New GEneration*, ANGE), l'implantation de l'architecture ODD/SAMURAAIE est un moteur de déploiement automatique. Il se présente sous la forme d'une bibliothèque comportant deux parties distinctes. La première implante le modèle SAMURAAIE d'abstraction des systèmes informatiques, et la seconde le modèle ODD d'automatisation de déploiement d'applications. La seconde partie utilise la première. Cette implantation comporte 3214 lignes uniques de code (*Single Lines Of Code*, SLOCs). Le langage de programmation de cette bibliothèque, Python [97], est orienté objet, typé dynamiquement, et interprété. Les dépendances entre les classes de la seconde partie sont injectées hors de leur définition, dans celle de la bibliothèque, selon le patron de conception « injection de dépendances » [53].

Le code source d'ANGE est conforme aux conventions Python et comporte des tests ainsi qu'une documentation embarquée. La conformité du code a été testée avec l'outil de validation statique `pylint` [98]. En plus des traditionnels tests unitaires, certains tests sont embarqués dans la documentation grâce aux conventions `doctest` de Python. Cette documentation est elle-même embarquée dans le code source grâce à la fonctionnalité `docstrings` de Python. Une archive de ce code est disponible sur demande.

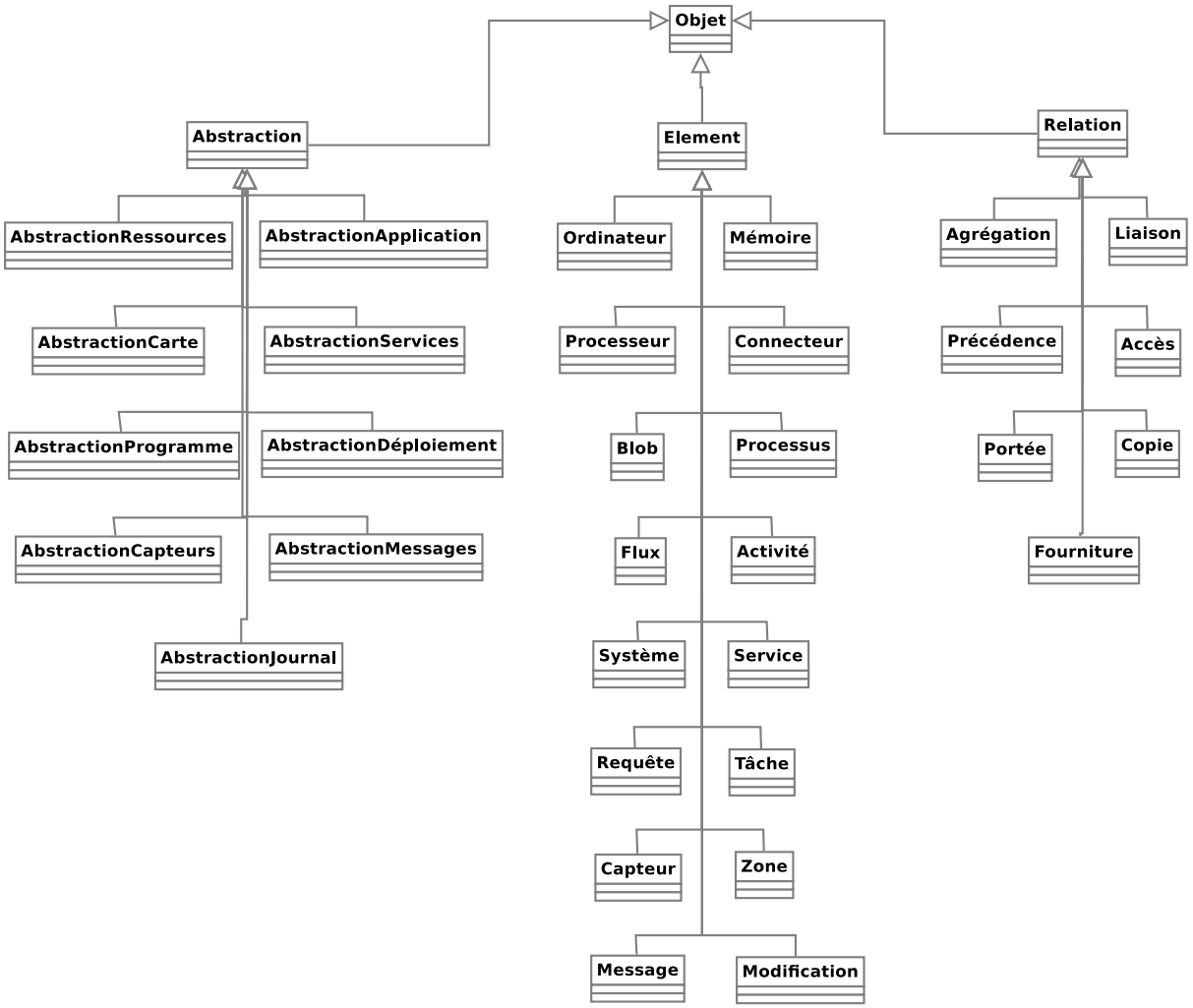


FIGURE 5.1 – Hiérarchie des classes d'objets de l'implantation du modèle SAMURAAIE

5.3 Implantation de SAMURAAIE

Cette section couvre trois volets de l'implantation du modèle SAMURAAIE. Le premier concerne les abstractions, les éléments, et les relations SAMURAAIE. Le deuxième concerne les caractéristiques des éléments et des relations. Le troisième et dernier volet concerne la sauvegarde de ces caractéristiques dans une base relationnelle de données. L'implantation du modèle SAMURAAIE comporte 945 lignes uniques de code, soit 29% de l'implantation complète.

5.3.1 Abstractions, éléments, et relations

La première partie implante les abstractions SAMURAAIE. Ces abstractions comportent des éléments et des relations entre ces éléments. Les éléments ont des caractéristiques, les relations ont un type. À ces abstractions correspondent des graphes orientés, aux éléments des sommets, et aux relations des arcs. Dans l'implantation, ces graphes, leurs sommets, et leurs arcs sont des classes d'objets. La figure 5.1 page 102 illustre la hiérarchie de ces classes d'objets. Afin d'abstraire l'environnement avec cette implantation, il suffit donc de créer des objets Abstraction et de leur ajouter des objets Élément et Relation.

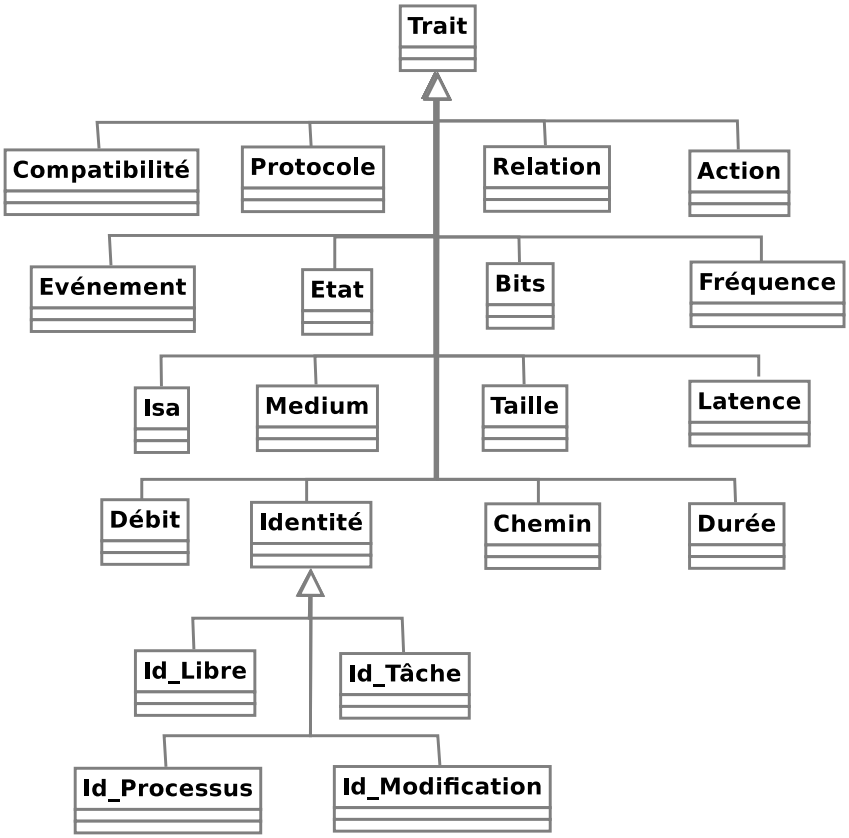


FIGURE 5.2 – Hiérarchie des traits des objets de l'implantation du modèle SAMURAAIE

5.3.2 Caractéristiques des éléments et des relations

Les caractéristiques des éléments et des relations des abstractions SAMURAAIE sont typées et les valeurs de ces caractéristiques peuvent changer. Dans l'implantation, ces caractéristiques ne sont pas directement des attributs des instances des classes Élément et Relation, mais des traits de ces objets. En effet, fondés sur un principe de meta-programmation, les traits d'objets permettent de mieux maîtriser les changements de leurs valeurs que les attributs d'objets. La figure 5.2 page 103 illustre la hiérarchie de ces traits d'objets. L'entreprise Enthought Incorporated développe et distribue une bibliothèque qui fournit le support des traits d'objets, c'est la bibliothèque utilisée.

5.3.3 Sauvegarde des caractéristiques dans une base relationnelle

Les caractéristiques des éléments et des relations des abstractions SAMURAAIE doivent persister après leur manipulation, leur volume peut dépasser la capacité en mémoire vive d'un ordinateur ordinaire, et elles doivent pouvoir être interrogées facilement. Dans l'implantation de SAMURAAIE, ces caractéristiques sont sauvegardées dans des tables d'une base relationnelle de données, et cette sauvegarde est automatisée par un associateur objet-base relationnelle (*Object-Relational Mapper*, ORM).

Le mécanisme de sauvegarde automatique est ajouté à la hiérarchie des classes de l'implantation de SAMURAAIE. La bibliothèque `SQLAlchemy` fournit à la fois l'associateur objet-base relationnelle et les adaptateurs de systèmes usuels de gestion de base de données. Cette sauvegarde des caractéristiques des éléments et des relations SAMURAAIE utilise au moins les deux patrons de conception suivants : « objet d'association » [99] et « héritage d'objets par jointure

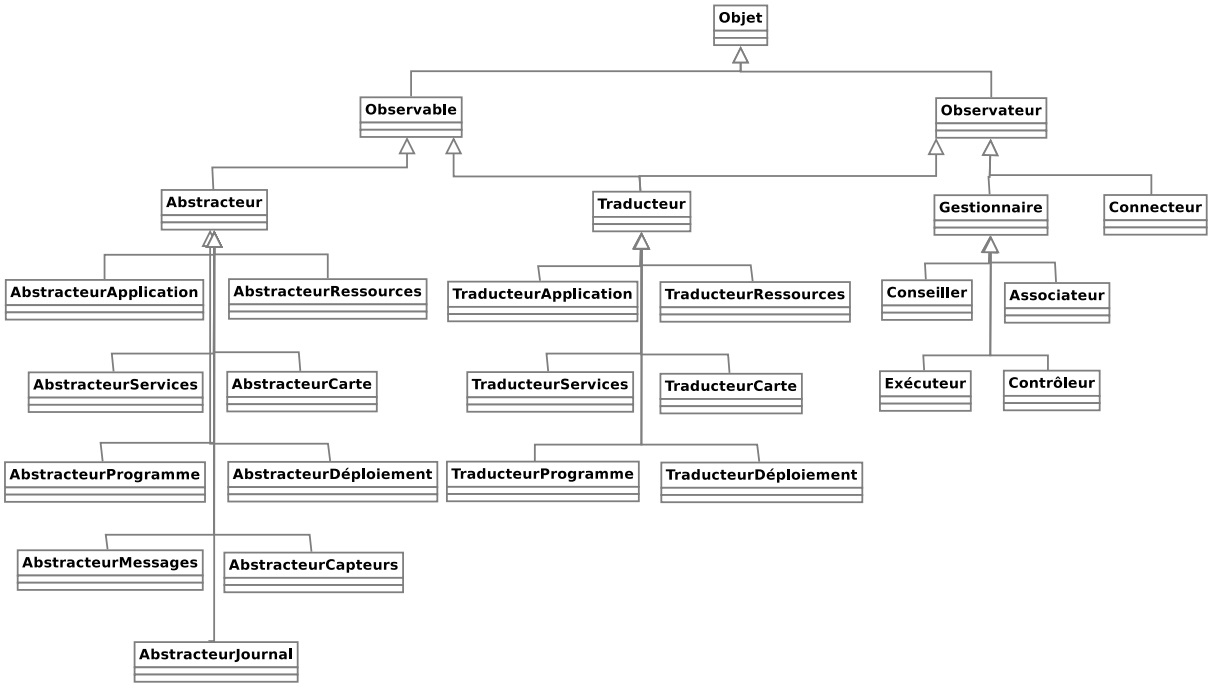


FIGURE 5.3 – Hiérarchie des classes d'objets de l'implantation du modèle ODD

de tables » [100].

À une relation d'éléments SAMURAAIE correspondent un arc incident à deux sommets SAMURAAIE, cet arc ayant une caractéristique, son type. En relationnel, une solution est que la table dans laquelle ces arcs sont sauvegardés ait autant de colonnes que d'attributs d'arc, et deux colonnes faisant référence à la table dans laquelle sont sauvegardés les sommets, comme dans une relation d'association. On peut en effet concevoir un arc comme étant un objet d'association entre sommets.

Les caractéristiques des éléments SAMURAAIE varient à la fois en type et en nombre. Or en relationnel, le type et le nombre des colonnes d'une table ne peut pas varier. Une solution est que les caractéristiques des éléments d'un niveau hiérarchique donné soient sauvegardés dans une table donnée, que seuls les caractéristiques supplémentaires des éléments du niveau inférieur soient sauvegardés dans une autre table, et enfin que cette autre table fasse référence à la première. Toutes les caractéristiques des éléments du niveau inférieur peuvent alors être regroupés, en faisant la jointure des deux tables.

5.4 Implantation d'ODD

La seconde partie implante les acteurs ODD. Ces acteurs s'appellent selon les règles du modèle de programmation orienté objets et s'observent selon le patron de conception « observateur » (*observer design pattern*) [101]. Les gestionnaires et les traducteurs observent les abstracteurs, les connecteurs observent les traducteurs. Les notifications comportent les références aux modifications, figurant au niveau Événements, ayant provoqué ces notifications. Dans l'implantation, ces acteurs sont des classes d'objets. La hiérarchie de ces classes comporte au moins deux niveaux : les interfaces d'acteurs et les acteurs, voire des spécialisations d'acteurs (voir la figure 5.3 page 104). Afin d'activer le moteur avec cette implantation, il suffit donc de créer ces objets acteurs et d'injecter les dépendances entre eux, selon l'ordre d'observation. Cette observation est séquentielle. L'implantation du modèle ODD comporte 2229 lignes uniques de code, soit 69% de

l'implantation complète.

5.4.1 Gestionnaires

Les gestionnaires ODD gèrent le déploiement en permanence. Associateur, Conseiller, Exécuteur, et Contrôleur interagissent avec les abstracteurs afin d'accéder aux abstractions. L'implantation des gestionnaires comporte 494 lignes uniques de code, soit 22% de l'implantation du modèle ODD et 15% de l'implantation complète.

Dans cette implantation, Associateur est l'objet dont la classe comporte le plus grand nombre de lignes uniques de code (318). Il implante les algorithmes d'association afin de modifier les abstractions SAMURAAIE Carte, Déploiement, et Journal. Ces algorithmes tiennent compte des modifications précédentes (algorithmes incrémentaux). Afin de modifier l'abstraction Carte, l'algorithme est récursif. En cas d'échec d'association pour cette abstraction, l'algorithme ajoute des copies de fichiers dans l'abstraction Application. L'algorithme modifiant l'abstraction Déploiement détecte les problèmes de portée des actions mais ne les résout pas.

Les classes Conseiller, Exécuteur, et Contrôleur sont moins développées que Associateur. Conseiller implante l'algorithme de transformation de l'abstraction Carte en abstraction Programme, et modifie cette dernière. Cet algorithme tient compte des modifications précédentes (algorithme incrémental). Exécuteur implante l'algorithme d'exécution des actions afin de modifier l'abstraction Déploiement. Cet algorithme tient compte des relations de précédence entre les actions, relations figurant dans l'abstraction Programme. Contrôleur implante l'algorithme de traitement des modifications des abstractions d'instance et d'action et modifie celles-ci. Par ailleurs cet algorithme supprime les messages et modifications SAMURAAIE qu'il a précédemment traités.

5.4.2 Abstracteurs

Les abstracteurs ODD offrent une vue cohérente sur les abstractions SAMURAAIE, et accèdent directement à ces dernières. Dans cette implantation, ces abstracteurs, c'est-à-dire les abstracteurs de ressources, d'application, de carte, de services, de programme, de déploiement, de capteurs, de messages, et de journal, manipulent leurs abstractions à travers les méthodes de celles-ci. Ces méthodes d'interrogation et de modification utilisent respectivement des requêtes en langage standard de requête (*Standard Query Language*, SQL) et des transactions. Les abstracteurs ne modifient cependant pas directement les abstractions d'instance et d'action SAMURAAIE, ils créent et ajoutent les messages, les modifications, et et les relations d'agrégation et de portée à l'abstraction de messages. L'implantation des abstracteurs comporte 384 lignes uniques de code, soit 17% de l'implantation du modèle ODD et 12% de l'implantation complète.

5.4.3 Traducteurs

Les traducteurs ODD traduisent l'information technologique en abstractions SAMURAAIE, et réciproquement. Dans cette implantation, ces traducteurs fournissent des méthodes d'interrogation et de modification de leurs abstractions, selon les interfaces des logiques des technologies qui les caractérisent. Ces méthodes utilisent celles des abstracteurs, à base de requêtes SQL et de transactions. En effet, comme les gestionnaires, les traducteurs accèdent indirectement à leurs abstractions, c'est-à-dire seulement à travers les abstracteurs. L'implantation des traducteurs comporte 987 lignes uniques de code, soit 44% de l'implantation du modèle ODD et 31% de l'implantation complète.

Cette implantation fournit d'emblée plusieurs traducteurs. Pour les abstractions de ressources, d'application, et de carte, elle fournit des traducteurs au format de document « en-

core un autre langage de balisage » (*Yet Another Markup Language*, YAML). Dans une syntaxe simple et lisible, le format utilisé permet de décrire l'ordinateur local et les ordinateurs distants, leurs interconnexions, des fichiers et répertoires, et les emplacements de ceux-ci sur les ordinateurs. Pour les abstractions de services, de programme, et de déploiement, cette implantation fournit des traducteurs POSIX. Enfin, elle fournit des traducteurs au format DOT pour toutes les abstractions afin de figurer celles-ci sous forme de graphes orientés avec les symboles du modèle SAMURAAIE.

5.4.4 Connecteurs

Les connecteurs ODD connectent les technologies de l'environnement à celle de l'implantation d'ODD, et réciproquement. Les implantations de ces connecteurs diffèrent donc d'une technologie à une autre. La première connexion que propose cette implantation est une console afin de suivre le fonctionnement de cette implantation. Outre cette connexion directe, en Python, cette implantation fournit d'emblée des connecteurs POSIX, POSIX par SSH, et YAML-RPC. L'implantation de ces connecteurs comporte 100 lignes uniques de code, soit 4% de l'implantation du modèle ODD et 3% de l'implantation complète.

Les connecteurs POSIX et POSIX par SSH permettent d'utiliser des ordinateurs, locaux ou distants, exploités par des systèmes GNU/Linux et sur lesquels des serveurs SSH attendent les connexions des utilisateurs. En effet, lorsque des traducteurs ajoutent de nouveaux ordinateurs et les systèmes POSIX qui les exploitent, le connecteur de services POSIX par SSH s'y connecte puis y installe et y exécute un agent au nom de l'utilisateur. Ces agents attendent la réception d'instructions, telles que la soumission de commandes ou le transfert de fichiers. Ce mécanisme d'auto-déploiement repose sur la bibliothèque d'exécution à distance de code Python, appelée `execnet`, que développe et distribue l'entreprise CodeSpeak.

Le connecteur YAML-RPC permet à des applications utilisant cette technologie de se connecter à distance à ANGE. YAML-RPC est une proposition de protocole d'appel à distance de procédures, dont les messages sont formatés dans le standard YAML. Pour cette technologie, le moteur ANGE est exécuté dans une forme de serveur et des clients peuvent alors l'appeler à distance. Présentée dans un chapitre précédent, parmi les technologies d'application, CORBA, l'architecture commune de courtier en requêtes d'objets, est un des standards mondiaux en programmation répartie à base d'objets.

5.5 Ajout du support d'une technologie

ANGE a vocation à supporter de front une multitude de technologies. Qu'il s'agisse de formats de description ou de systèmes d'information à propos des ressources, de systèmes ou d'intergiciels d'exploitation, de plates-formes d'applications, ou d'interfaces utilisateur, ANGE est conçu pour les accueillir. L'ajout à ANGE du support d'une technologie comporte deux étapes. La première est d'implanter les connecteurs et les traducteurs pour cette technologie. La seconde est d'ajouter ces connecteurs et traducteurs dans la bibliothèque.

Dans le modèle ODD d'automatisation du déploiement d'applications, les connecteurs constituent une double interface de programmation et les traducteurs transforment l'information. Les interfaces externes des connecteurs sont libres mais il est recommandé qu'elles soient les plus conformes possibles à la logique de la technologie considérée. Par exemple, un connecteur d'application à base de composants devrait être un composant. Il faut veiller à ce que des traducteurs de déploiement puissent appeler les interfaces internes, sinon un nouveau traducteur de déploiement devra également être écrit. Une fois les connecteurs implantés, l'information à transformer est généralement connue et implanter les traducteurs est alors direct.

Implantation	SLOCs en %
samuraaie	29
odd	69
odd.managers	15
odd.abstracters	12
odd.translators	31
odd.connectors	3

TABLE 5.1 – Répartition du code source

Une fois les classes Connecteur et Traducteur spécialisées en autant de classes de connecteurs et de traducteurs nécessaires pour supporter la technologie considérée, il reste à déclarer ces implantations. La bibliothèque ANGE comporte une partie d'initialisation dans laquelle se trouve un registre d'objets à partir duquel sont injectées les dépendances entre les classes d'objets. Il s'agit donc d'ajouter les nouvelles implantations à ce registre, en spécifiant les autres entrées dont elles auraient besoin. Par exemple, les traducteurs auront certainement besoin d'abstracteurs, des dépendances avec ces derniers devront donc être injectées.

5.6 Discussion

ANGE est une première implantation de l'architecture ODD/SAMURAAIE de déploiement dynamique d'application. Sous forme de moteur centralisé, cette implantation est directe et extensible. Par ailleurs la séparation des préoccupations entre les différentes classes d'acteurs est respectée. En effet, le gestionnaire Exécuteur ne communique par exemple avec aucun connecteur et seul les traducteurs de déploiement appellent les connecteurs de services et de programme.

Parce qu'il utilise le modèle d'acteurs [37], le modèle ODD est conçu pour être implanté de manière concurrente. De même, les éléments des abstractions SAMURAAIE pourraient être répartis sur plusieurs ordinateurs, selon les zones délimitées par les portées des capteurs. Pour le modèle d'acteurs, un langage de programmation l'implantant nativement aurait pu être utilisé, Erlang par exemple. L'usage d'une bibliothèque implantant le modèle d'acteurs comme une extension du langage Python, appelée PARLEY [102], a été envisagé mais celle-ci ne supportait pas la répartition sur plusieurs ordinateurs et n'existait alors qu'en version beta.

Le code source d'ANGE se découpe en trois parts presque égales (voir le tableau récapitulatif 5.1 page 107). La première est l'implantation du modèle SAMURAAIE. La deuxième est l'implantation des traducteurs ODD. La troisième et dernière part est l'implantation du reste du modèle ODD. Cette répartition suggère à la fois que l'implantation du modèle SAMURAAIE est directe, que celle du mécanisme de traduction est plus exigeante, et que celle du reste du modèle ODD doit être enrichie.

Dans cette première implantation, les algorithmes des gestionnaires Associateur et Conseiller sont naïfs. Plusieurs techniques permettraient de les améliorer. Ils pourraient comporter des heuristiques, être distribués sur plusieurs ordinateurs, être remplacés par des meta-heuristiques, par un solveur de problème de satisfaction de contraintes, ou par un système expert. Dans ce domaine, des résultats en recherche opérationnelle peuvent ainsi être appliqués afin de perfectionner la logique de déploiement.

Le connecteur fourni d'emblée par cette première implantation est limité. En particulier, l'agent des connecteurs POSIX et POSIX par SSH savent effectuer les instructions que leur envoie la partie centrale du connecteur, ils ne savent cependant pas retransmettre ces instructions à d'autres agents lorsque celles-ci ne leur sont pas destinées. Bien que l'architecture ODD/SAMURAAIE le permette, ceci empêche ANGE d'envoyer des instructions au-delà des

ordinateurs auxquels la partie centrale est directement connectée. Par ailleurs, davantage de connecteurs sont attendus, en particulier pour les ordinateurs exploités par des gestionnaires de lots ou pour les intergiciels d'applications MPI, CCM/CORBA, et en flot de travail.

5.7 Conclusion

En moins de 3300 lignes uniques de code, le moteur ANGE est la première implantation de l'architecture ODD/SAMURAAIE de déploiement dynamique d'applications. L'association objet-base relationnelle rend les abstractions SAMURAAIE persistantes et permet d'interroger facilement celles-ci. L'injection des dépendances entre objets, qui a lieu lors de l'initialisation du moteur, facilite l'ajout du support de davantage de technologies. Ce support prend simplement la forme de spécialisations des classes Traducteur et Connecteur. A priori, des implantations réparties des abstractions et des acteurs étant possibles, la taille des infrastructures et de l'application supportées ne semble pas limitée.

Bien que des améliorations soient déjà envisagées, cette première implantation va permettre de mener, dans un chapitre suivant, les expériences qui valideront pratiquement l'architecture ODD/SAMURAAIE proposée dans le chapitre précédent.

Quatrième partie

Validation

Chapitre 6

Automatisation du déploiement d'applications parallèles, paramétriques, et en flot de travail

6.1 Introduction

Les deux chapitres précédents proposent respectivement ODD/SAMURAAIE, une architecture de déploiement dynamique d'application, et ANGE, un prototype de cette architecture. Grâce au modèle SAMURAAIE sur lequel il se fonde, le modèle ODD peut automatiser différents cas de déploiement pour l'utilisateur. Cette architecture est particulièrement destinée aux cas de déploiement impliquant des technologies parallèles, hétérogènes, et dynamiques. Quant à ANGE, il a été développé afin de prouver le concept de cette architecture.

Ce chapitre met l'architecture ODD/SAMURAAIE à l'épreuve du déploiement automatique de plusieurs modèles d'applications sur plusieurs architectures de ressources. Il commence par présenter les principes de cette épreuve et par motiver des choix de modèles d'applications et d'architectures de ressources. Ensuite, il détaille successivement chacun des cas de déploiement, en évaluant l'architecture. Enfin, il discute les résultats de cette épreuve.

6.2 Principes

Le protocole d'évaluation de l'architecture ODD/SAMURAAIE comporte trois parties. Il s'agit d'abord de fixer trois scénarios permettant de couvrir progressivement la plupart des cas de déploiement dans lesquels un utilisateur puisse se trouver. Chaque scénario comporte une architecture de ressources, un modèle d'applications, et surtout des instances précises de ceux-ci. Ces scénarios partagent le même objectif : l'utilisateur délègue les préoccupations du processus de déploiement. La deuxième partie confronte l'architecture avec chaque scénario. Il s'agit d'expliquer comment le scénario est abstrait avec SAMURAAIE puis comment il est automatisé avec ODD. La troisième et dernière partie de cette évaluation consiste à discuter les résultats obtenus.

Le choix des architectures de ressources et de leurs instances porte sur un ordinateur, une grappe d'ordinateurs, et une fédération de grappes d'ordinateurs, tous distincts de l'ordinateur personnel de l'utilisateur. Le choix de l'ordinateur permet d'évaluer le déploiement sur des ressources distantes et immédiatement exploitables. Le choix de la grappe permet d'évaluer le déploiement sur des ressources à la fois parallèles et exploitées en différé. En supposant que des ordinateurs soient libérés ou récupérés pendant l'exécution de l'application, une grappe est éga-

lement une architecture dynamique. Enfin, le choix de la fédération de grappes permet d'évaluer le déploiement sur des ressources parallèles et dynamiques et hétérogènes.

Le choix des modèles d'applications et de leur instances porte sur une application parallèle, une application paramétrique, et une application en flot de travail. Le choix de l'application parallèle est trivial. Le choix de l'application paramétrique, également qualifiée de maître-esclaves, permet d'évaluer le déploiement d'applications parallèles. En supposant que le nombre de ressources allouées à cette application varie, le modèle d'application paramétrique est également dynamique. Enfin, le choix de l'application en flot de travail permet d'évaluer le déploiement d'applications parallèles et dynamiques. En supposant que les tâches de cette application utilisent des technologies différentes, le modèle d'applications en flot de travail est également hétérogènes.

Pour chaque scénario, l'utilisateur considéré dispose d'un ordinateur personnel, tel qu'une station de travail ou bien un ordinateur portable. Cet ordinateur est interconnecté avec les ressources. Sur cet ordinateur se trouvent le conditionnement des applications ainsi que l'implantation de l'architecture ODD/SAMURAAIE.

6.3 Déploiement sur ordinateurs d'applications parallèles

Dans cette section, l'utilisateur a accès à un ordinateur multi-processeur. Il veut y déployer une application parallèle dont il dispose. Après avoir détaillé l'ordinateur et l'application, cette section explique comment l'architecture ODD/SAMURAAIE traite ce cas de déploiement.

6.3.1 Un ordinateur et une application parallèle

L'ordinateur considéré est une station multi-processeur de travail, exploitée par GNU/Linux. Elle comporte deux processeurs AMD 64 bits ayant chacun quatre cœurs. Elle comporte également 8 Go de mémoire vive et deux disques durs SATA, en RAID0, sur lesquels se trouvent le répertoire `/home/user` personnel de l'utilisateur ainsi qu'un répertoire `/local` temporaire, répertoires de 100 Go chacun. Cette station est connectée à un réseau local TCP/IP sur Ethernet Gigabit (par seconde), réseau dans lequel elle est appelée `computer`. Elle accepte des commandes par SSH et des transferts de fichiers par SCP.

L'application considérée est une application multi-processus, compilée avec une implantation de MPI faisant communiquer ces processus par mémoire partagée. Compilée pour GNU/Linux AMD 64 bits, cette application comporte les trois fichiers suivants :

- Le premier fichier est le programme, un binaire exécutable faisant appel à une bibliothèque partagée. Ce programme accepte deux paramètres : le nombre de processus et le chemin d'accès à un jeu de données à traiter. Il écrit systématiquement ses résultats dans des fichiers `/local/output.n` où n est le rang du processus parmi tous les processus exécutés parallèlement. La taille d'un fichier de résultats est toujours au moins 100 fois inférieure à celle du jeu de données à traiter.
- Le deuxième est un fichier binaire de bibliothèque partagée. Il contient l'implantation des fonctions d'intercommunication par mémoire partagée des processus.
- Le troisième et dernier fichier est un jeu de données à traiter, dont la taille est de 500 Mo.

L'utilisateur a ouvert une session de travail sur son ordinateur personnel. Il a un compte sur l'ordinateur distant, ordinateur qui fonctionne sans que personne d'autre ne l'utilise. Ces deux ordinateurs sont interconnectés au sein du même réseau local. Les fichiers de son application se trouvent dans un sous-répertoire `appli` de son répertoire personnel sur son ordinateur. Enfin, il ne sait pas quel nombre de processus exécuter parallèlement pour son application.

6.3.2 Traitement avec ODD/SAMURAAIE

Cette sous-section détaille d'abord, abstraction par abstraction, comment le scénario de déploiement sur un ordinateur d'une application parallèle est abstrait avec le modèle SAMURAAIE. Elle détaille ensuite, acteur par acteur, comment ce scénario est automatisé avec le modèle ODD.

Abstraction avec SAMURAAIE

Ressources L'ordinateur personnel de l'utilisateur, la station de travail à laquelle il a accès, et leur interconnexion sont abstraits dans l'abstraction Ressources.

L'ordinateur personnel est un élément Ordinateur agrégeant deux éléments Mémoire dont une vive et une de masse, un élément Processeur, et $2*2 = 4$ éléments Connecteur. $4*2 = 8$ relations d'accès relient, en lecture et en écriture, les éléments Processeur et Mémoire aux éléments Connecteur. Tous les éléments ont l'état `OK`. En l'occurrence, les valeurs de leurs caractéristiques sont indifférentes au traitement du cas de déploiement considéré. Seuls les média des mémoires, la capacité de la mémoire de masse, et le chemin d'accès des connecteurs permettant au processeur de lire et d'écrire dans cette mémoire sont fixés : respectivement `RAM` et `HDD`, 10 Go, et deux fois `/home/user`.

La station de travail est un élément Ordinateur agrégeant trois éléments Mémoire dont une vive et deux de masse, huit éléments Processeur, et $3 * 2 = 6$ éléments Connecteur. $8 * 6 + 6 = 54$ relations d'accès relient, en lecture et en écriture, les éléments Processeur et Mémoire aux éléments Connecteur. De même que ceux de l'ordinateur personnel, tous les éléments de cette station ont l'état `OK`. En revanche, les valeurs de leurs caractéristiques importent. Les mémoires de masse, correspondant non pas aux disques durs en RAID0 mais aux deux répertoires sur ce couple de disques, ont chacune une capacité de 100 Go. Le débit des connecteurs permettant des accès en lecture et en écriture dans ces mémoires se partagent le débit du couple de disques. Le chemin d'accès de ces connecteurs est `/home/user` et `/local`. Au partage des mémoires par les deux processeurs quadri-cœur correspond la concentration de $6 * 8 = 48$ relations d'accès, reliant les éléments Processeur aux éléments Connecteur, et $2 * 3 = 6$ (huit fois moins) autres relations d'accès reliant ces éléments Connecteur aux éléments Mémoire.

L'interconnexion de l'ordinateur personnel avec la station de travail est abstraite par deux éléments Connecteur et onze relations d'accès. Ces relations relient, en écriture seulement, les éléments Processeur de l'ordinateur personnel et Mémoire vive de la station au premier élément Connecteur, d'une part, et les éléments Processeur de la station et Mémoire de l'ordinateur personnel au second élément Connecteur, d'autre part. Ces éléments Connecteur ont l'état `OK`. Leurs latences, débits, protocoles, et adresses sont ceux de l'interconnexion. Par exemple, la valeur de l'adresse du premier élément Connecteur est `computer`, c'est-à-dire l'identifiant sur le réseau de la station. L'élément Processeur de l'ordinateur personnel écrit donc directement dans l'élément Mémoire de la station, et réciproquement.

Application L'implantation de l'architecture ODD/SAMURAAIE, le répertoire des fichiers de l'application, son processus, et ses fichiers de résultats sont abstraits dans l'abstraction Application. Le connecteur ODD d'application MPI fixe le nombre n avant de demander au traducteur d'enregistrer l'application auprès de l'abstracteur correspondant, avant l'étape d'allocation du processus de déploiement. En l'occurrence, ce nombre est 8. Le répertoire des fichiers de l'application est un élément Blob. Cet élément a l'état `TERMINÉ`.

L'implantation de l'architecture ODD/SAMURAAIE est un processus particulier puisque, en quelque sorte, il s'abstrait lui-même. Cette abstraction est nécessaire afin de déterminer la localité, dans le réseau, de l'ordinateur sur lequel cette implantation est exécutée. Il s'agit

d'un élément Processus agrégeant un élément Blob d'empreinte en mémoire vive, un élément Activité, et deux éléments Flux. Quatre relations d'accès relient, en lecture et en écriture, les éléments Blob et Activité aux éléments Flux. Tous les éléments ont l'état `OK`. Les valeurs de leurs caractéristiques sont indifférentes au traitement du cas de déploiement considéré.

Les huit processus de l'application sont des éléments Processus agrégeant d'abord l'élément Blob du répertoire des fichiers de l'application, huit éléments Blob d'empreinte en mémoire vive, un élément Blob de mémoire partagée, huit éléments Blob de fichiers de résultats, huit éléments Activité, et $3 * 8 + 3 = 27$ éléments Flux. $9 * 8 + 3 = 75$ relations d'accès relient, en lecture, en écriture, ou les deux, les éléments Blob et Activité aux éléments Flux. Contrairement à ceux des ressources, ces éléments ont d'abord l'état `ATTENDU`, puis `OK`, puis `TERMINÉ`. Les tailles des fichiers de résultats sont 100 fois inférieures à celle du répertoire des fichiers de l'application. L'architecture de jeu d'instruction des activités est AMD et leur nombre de bits est 64. Le chemin d'accès des flux permettant aux activités d'écrire dans les fichiers de résultats est `/local/output.n` où n est le rang du processus. Enfin, la ligne de commande d'un des processus est `./a.out 8 ./data.in` et celle des autres est vide.

À partir de l'étape d'allocation, des copies de fichiers sont ajoutées dans l'application. Le répertoire des fichiers de l'application est copié et cette copie remplace l'original dans les processus de l'application. De même, les fichiers de résultats sont copiés. Ces copies sont des éléments Blob qui sont reliés aux originaux par des relations de copie.

Carte La présence du répertoire de l'application puis des fichiers de résultats, sur l'ordinateur personnel de l'utilisateur, et celle des processus de l'application, sur la station de travail à laquelle cet utilisateur a accès, sont abstraites dans l'abstraction Carte.

La présence du répertoire `appli` des fichiers de l'application dans le répertoire `/home/user` de l'utilisateur sur son ordinateur est une relation de liaison entre les éléments Blob et Mémoire correspondants. À partir de l'étape d'allocation du processus de déploiement, une copie de ce répertoire `appli` est présente dans le répertoire `/home/user` de l'utilisateur sur la station. Cette présence est également une relation de liaison entre les éléments Blob et Mémoire correspondants.

Les fichiers de résultats sont d'abord attendus dans le répertoire personnel de l'utilisateur sur son ordinateur. Des relations de liaison relient alors les éléments Blob et Mémoire correspondants. Or, les processus qui les écrivent ne pouvant pas être exécutés sur cet ordinateur, ces relations sont détruites puis reconstruites avec les éléments de copie de ces fichiers de résultats. Les originaux, eux, sont alors présents sur le répertoire `/local` de la station. Des relations de liaison relient alors les éléments Blob et Mémoire correspondants.

À partir de l'étape d'allocation du processus de déploiement, les processus de l'application sont associés à la station. Des relations de liaison relient alors les éléments Processus, Blob, Activité, et Flux aux éléments Ordinateur, Mémoire, Processeur, et Connecteur correspondants.

Services Le système d'exploitation de l'ordinateur personnel de l'utilisateur, celui de la station de travail à laquelle il a accès, et leurs services de mémorisation et d'exécution sont abstraits dans l'abstraction Services.

Le système d'exploitation de l'ordinateur personnel est un élément Système agrégeant trois éléments Service de mémorisation. Ces services permettent de créer, copier, ou supprimer des fichiers dans le répertoire personnel de l'utilisateur, tels sont donc leurs types d'action. Quatre relations de portée relient l'élément Système à l'élément Ordinateur de l'utilisateur, et les éléments Service à l'élément Mémoire de son répertoire personnel. Tous ces éléments ont l'état `OK`.

Le système d'exploitation de la station est également un élément Système. Cet élément agrège sept éléments Service de mémorisation et d'exécution. Ces services permettent de créer, copier,

et supprimer des fichiers dans les mémoires de masse de la station, et de démarrer, suspendre, reprendre, ou arrêter des activités sur ses processeurs. $3 * 2 + 1 + 4 * 8 = 39$ relations de portée relient les éléments Service de mémorisation aux éléments Mémoire de masse de la station, l'élément Service de copie de fichiers également à l'élément Mémoire de masse de l'ordinateur de l'utilisateur, et les éléments Service d'exécution aux éléments Processeur de la station. Tous ces éléments ont l'état `OK`.

Programme Le flot de requêtes de services de mémorisation et d'exécution est abstrait dans l'abstraction Programme. Ce flot est un élément Tâche agrégeant $1 + 8 + 8 = 17$ éléments Requête. Ces requêtes demandent les actions suivantes :

- Copier le répertoire des fichiers de l'application, depuis le répertoire personnel de l'utilisateur sur son ordinateur, vers celui de la station ;
- Démarrer les activités de l'application sur les processeurs de la station ;
- Copier les fichiers de résultats, depuis le répertoire temporaire de la station, vers le répertoire personnel de l'utilisateur sur son ordinateur.

17 relations de portée et $8 + 8 = 16$ relations de précédence relient ces éléments Requête. Les relations de portée relient les éléments Requête aux éléments Blob des copies et Activité. Les relations de précédence relient l'élément Requête de copie du répertoire des fichiers de l'application aux éléments Requête de démarrage des activités. Elles relient également chacun de ces derniers à l'élément Requête de copie du fichier de résultats correspondant à l'activité dont il demande le démarrage. Avant de se terminer, tous ces éléments ont l'état `OK`. Auparavant, ils ont l'état `ATTENDU`.

Déploiement L'association des requêtes de services et des services eux-mêmes est abstraite dans l'abstraction Déploiement. À cette association correspondent $1 + 1 + 8 + 8 = 18$ relations de liaison reliant les éléments Tâche et Requête aux éléments Système et Service portant sur la station de travail.

Capteurs, Messages, et Journal Les changements survenant pendant le traitement du cas de déploiement considéré concernent les trois étapes suivantes :

Enregistrement Ces changements enregistrent d'abord l'ordinateur de l'utilisateur, l'implantation de l'architecture ODD/SAMURAAIE, la présence de cette implantation dans cet ordinateur, la station de travail, l'interconnexion, les systèmes d'exploitation et leurs services, le répertoire des fichiers de l'application, et la présence du répertoire des fichiers de l'application dans l'ordinateur de l'utilisateur.

Demande de déploiement Ces changements enregistrent ensuite les processus de l'application, les fichiers de résultats de celle-ci, la présence de ces fichiers dans le répertoire personnel de l'utilisateur sur son ordinateur, et mettent ces processus et ces fichiers en attente.

Déploiement Ces changements enregistrent des copies du répertoire des fichiers de l'application et des fichiers de résultats, remplacent ce répertoire par sa copie dans les processus de l'application, associent ces processus à la station de travail, ajoutent la tâche d'installation et d'exécution de l'application, associent cette tâche au système d'exploitation de la station, changent l'état de cette tâche afin qu'elle soit réalisée.

Les changements apportés aux abstractions Ressources, Application, Carte, Services, Programme, et Déploiement sont abstraits dans les abstractions Capteurs, Messages, et Journal. Les zones modifiables des abstractions d'instance et d'action sont des éléments Zone agrégeant des éléments Capteur. En l'occurrence, chaque abstraction n'est couverte que par une seule zone. Des relations de portée relient chaque élément Capteur à tous les éléments de l'abstraction qu'il couvre. Un

changement est un élément Modification agrégeant des éléments Message. Des relations de portée relient ces messages à des éléments de la zone d'abstraction qu'il modifie. Des relations de liaison relient des capteurs à des messages et leurs zones à leurs modifications. Avant d'être traités, les messages ont l'état `ATTENDU`. Enfin, de même que pour une tâche, une modification ne peut avoir l'état `TERMINÉ` que quand tous les éléments qu'elle agrège ont également cet état.

Automatisation avec ODD

Gestionnaires

Associateur Le gestionnaire Associateur associe les éléments de contenu en attente avec des éléments de contenant fonctionnant correctement. Dans le cas de déploiement considéré, les premiers éléments de contenu en attente sont les fichiers de résultats de l'application. Ils sont cependant associés d'emblée au répertoire personnel de l'utilisateur sur son ordinateur. La première association que ce gestionnaire tente de réaliser est donc celle de l'application elle-même, c'est-à-dire de ses processus et de tous les éléments que ceux-ci agrègent.

Afin d'associer un élément d'agrégation, le gestionnaire Associateur part des éléments déjà associés que cet élément agrège. Quand il tente d'associer les processus de l'application, il part donc des fichiers de résultats. Or les activités qui écrivent ces fichiers sont incompatibles avec le processeur de l'ordinateur dans lequel ces fichiers sont attendus. En effet, la valeur d'architecture de jeu d'instructions et le nombre de bits que ces activités demandent sont AMD et 64 alors que ces caractéristiques ne sont pas précisées pour le processeur de l'ordinateur de l'utilisateur. Associateur cherche donc une autre solution.

Lorsque des associations de fichiers avec des mémoires empêchent l'association des processus avec les ordinateurs qui les agrègent, le gestionnaire Associateur ajoute des copies de ces fichiers et retente d'associer ces mêmes processus avec d'autres ordinateurs. Les fichiers de résultats ne pouvant pas être écrits par l'application directement sur l'ordinateur de l'utilisateur, Associateur ajoute deux types de copies dans l'application : une copie du répertoire des fichiers de l'application, et des copies des fichiers de résultats.

Afin d'ajouter les copies du répertoire des fichiers de l'application et des fichiers de résultats, le gestionnaire Associateur effectue plusieurs changements dans l'abstraction Application. Pour la copie du répertoire, il ajoute un élément Blob ayant les mêmes valeurs de taille et de médium que l'élément original, relie cette copie à l'original par une relation de copie, détruit les relations d'accès et d'agrégation de l'original avec les éléments Flux et Processus, et enfin reconstruit ces relations détruites pour la copie. Contrairement à l'original, dont l'état est `TERMINÉ` et qui reste associé avec la mémoire de masse de l'ordinateur de l'utilisateur, cette copie est d'abord en attente et n'est pas encore associée.

Afin d'ajouter les copies des fichiers de résultats, le gestionnaire Associateur détruit les relations de liaison des éléments originaux avec l'élément Mémoire du répertoire personnel de l'utilisateur sur son ordinateur, ajoute des éléments Blob de copie ayant les mêmes valeurs de taille et de médium que ces fichiers, relie ces copies aux originaux par des relations de copie, et enfin reconstruit les relations de liaison, avec la mémoire de l'ordinateur, détruites pour les copies. Contrairement au cas précédent de copie, ces copies sont associées et les originaux ne le sont pas encore.

Une fois ajoutées les copies, le gestionnaire Associateur retente et réussit l'association de tous les éléments de contenu en attente. Il relie donc, par des relations de liaison, l'élément Ordinateur de la station avec les éléments Processus de l'application, l'élément Mémoire du répertoire personnel de l'utilisateur sur la station avec l'élément Blob de copie du répertoire des fichiers de l'application, l'élément Mémoire vive de la station avec les éléments Blob des empreintes en mémoire vive de l'application, les éléments Processeur de la station avec les

éléments Activité de l'application, l'élément Mémoire du répertoire temporaire sur la station avec les éléments Blob des fichiers de résultats originaux, et les éléments Connecteur de la station avec les éléments Flux de l'application. En particulier, Associateur valide la compatibilité du chemin d'accès `/local` des connecteurs permettant aux processeurs de lire et d'écrire dans le répertoire temporaire de la station avec les chemins d'accès `/local/output.n` des flux permettant aux activités de l'application d'écrire les fichiers de résultats. Il valide également que les sommes des tailles des fichiers et des empreintes en mémoire vive restent inférieures aux capacités des mémoires avec lesquelles ces blobs sont associés.

Une fois associée l'application avec la station de travail, le gestionnaire Associateur attend que de nouveaux éléments de contenu passent en attente. Des éléments de contenu en attente, au niveau Actions, sont alors ajoutés dans l'abstraction Programme. Il s'agit de l'élément Tâche agrégeant des éléments Requête de services. Ces requêtes visent d'abord à copier le répertoire des fichiers de l'application, puis à en démarrer les activités, et finalement à en copier les fichiers de résultats. Associateur tente alors de les associer. En analysant les types d"action et les relations de portée de ces éléments Requête avec des éléments Blob et Activité, il les associe avec les éléments Système et Service portant sur la station de travail. En particulier, il valide que le service de copie de ce système porte effectivement sur les répertoires de la station mais également de l'ordinateur, afin que soient vouées au succès les requêtes de copie depuis l'ordinateur vers la station et réciproquement.

Le gestionnaire Associateur est également chargé d'associer les éléments Message en attente à des éléments Capteur fonctionnant correctement. De même que pour les associations au niveau Actions, Associateur doit analyser les types d'événement et les relations de portée de ces éléments afin de valider leur compatibilité. Une fois deux éléments de contenu et de contenant compatibles, il les relie par une relation de liaison qu'il ajoute dans l'abstraction Journal.

Conseiller Le gestionnaire Conseiller intervient dès que des relations de liaison relient, au niveau Instances, des éléments de contenu en attente avec des éléments de contenant. Sa première intervention concerne donc les fichiers de résultats. En effet, les éléments Blob de ces fichiers sont effectivement en attente et sont, dans un premier temps, associés avec l'élément Mémoire du répertoire personnel de l'utilisateur sur son ordinateur. En analysant les autres relations de ces éléments Blob de fichiers de résultats, Conseiller constate que ces fichiers ne sont pas des copies et que les flux et activités qui les écrivent ne sont qu'enregistrés. Il décide alors de mettre en attente ces flux et activités puis attend.

Le gestionnaire Conseiller intervient une seconde fois, afin de décider des actions de copie des fichiers et de démarrage des activités qui doivent l'être. Cette seconde intervention concerne donc la copie du répertoire des fichiers de l'application, les activités de l'application, et les copies des fichiers de résultats. Parmi tous les éléments de contenu en attente et associés avec des éléments de contenant, Conseiller s'intéresse d'abord aux fichiers ou répertoires seulement lus. Il ajoute ainsi une requête de copie portant sur la copie du répertoire des fichiers de l'application. Les requêtes de copie ne portent jamais sur un original car plusieurs copies peuvent avoir lieu à des moments différents : désigner l'original empêcherait cette distinction. Conseiller s'intéresse ensuite aux activités qui lisent le répertoire des fichiers de l'application, ajoute les requêtes de démarrage portant sur celles-ci, et relie ces requêtes avec celle de copie par des relations de précédence. Il s'intéresse ensuite aux fichiers que ces activités écrivent, constate qu'aucune autre activité ne les lit mais qu'ils sont reliés par des relations de copie. Il ajoute alors des requêtes de copie portant sur les copies de ces fichiers et les relie avec celles de démarrage des activités qui en écrivent les originaux par des relations de précédence. Enfin, il agrège toute ces requêtes dans une tâche et met le tout en attente.

Exécuteur Le gestionnaire Exécuteur déclenche les tâches en attente et les flots de requêtes que ces tâches comportent, afin qu'elles soient réalisées. Dans le cas de déploiement considéré, son intervention concerne la tâche de copie du répertoire des fichiers de l'application, le démarrage des activités de celle-ci, et la copie de ses fichiers de résultats. Après avoir attendu que de nouvelles actions soient décidées, Exécuteur constate effectivement que cette tâche est en attente et associée avec le système d'exploitation de la station de travail. Il analyse les relations de précédence des requêtes que cette tâche agrège et prévoit de déclencher la tâche ainsi que la requête qui n'est précédée d'aucune autre : la requête de copie du répertoire des fichiers de l'application. Il déclenchera ensuite toutes les requêtes qui succèdent directement à cette première requête : celles de démarrage des activités. Enfin, il déclenchera les dernières requêtes de la tâche : celles de copie des fichiers de résultats. Son intervention se terminera quand il aura marqué la tâche comme étant terminée, une fois que toutes les requêtes qu'elle comporte le seront.

En analysant les éléments à leur portée, le gestionnaire Exécuteur sait exactement quand déclencher des requêtes successives. L'important est de ne pas démarrer des activités avant que les fichiers qu'elles lisent ne leur soient lisibles, et de ne pas copier des fichiers avant que les activités qui les écrivent ne soient terminées. Avant de déclencher toutes les requêtes de démarrage des activités, requêtes qui succèdent directement à celle de copie du répertoire des fichiers de l'application, Exécuteur attend que l'élément Blob de copie de ce répertoire quitte ses états `ATTENDU` et `OK` pour l'état `TERMINÉ`. De même, avant de déclencher une requête parmi celles de copie des fichiers de résultats, requête qui succède directement à celle de démarrage de l'activité qui écrit le fichier de résultats considéré, Exécuteur attend que ce fichier quitte ses états `ATTENDU` et `OK` pour l'état `TERMINÉ`. Il s'assure ainsi que l'activité qui écrit ce fichier soit terminée mais pourrait également le vérifier.

Contrôleur Les interventions du gestionnaire Contrôleur concernent le traitement de tous les événements, elles sont donc très nombreuses. De même qu'Exécuteur au niveau Actions, au niveau Événements Contrôleur analyse les relations de précédence entre les différents éléments Modification et entre les différents éléments Message agrégés dans une même Modification. En revanche, contrairement à Exécuteur, non seulement Contrôleur déclenche des éléments en attente, mais il les traite. Pour ce faire, il en analyse le type d'événement et les relations de portée puis change directement les abstractions concernées.

Traducteurs Dans le cas de déploiement considéré, l'information traduite porte sur les ordinateurs et leur interconnexion, l'implantation de l'architecture ODD/SAMURAAIE, l'application MPI, la présence d'éléments de l'application sur des éléments des ressources (les « pseudo-chemins » SAMURAAIE), et le système GNU/Linux. À chacune de ces technologies correspond donc un traducteur ODD.

Traducteur d'ordinateurs Le traducteur d'ordinateurs traduit l'information technologique à propos d'ordinateurs interconnectés en éléments et relations entre éléments, pour l'abstraction Ressources. Cette information lui est fournie par un connecteur ODD, sous la forme d'un dictionnaire dont les clefs sont des noms de caractéristiques et les valeurs les valeurs de celles-ci. Il traduit également l'abstraction Ressources à des fins précises. En particulier, il accepte des valeurs d'architecture de jeu d'instructions et de nombre de bits et retourne le plus grand nombre de processeurs partageant une mémoire vive et ayant ces valeurs caractéristiques.

Le traducteur d'ordinateurs est appelé trois fois. La première fois, il s'agit d'abstraire l'ordinateur personnel de l'utilisateur. L'information transmise est alors quasiment nulle. Sans entrée dans le dictionnaire pour une caractéristique donnée, le traducteur s'appuie sur la structure d'un ordinateur simple et ne précise pas la valeur de cette caractéristique. La deuxième fois, il s'agit

d'abstraire la station de travail à laquelle l'utilisateur a accès ainsi que son interconnexion avec l'ordinateur personnel. L'information technologique est alors complète et précise notamment le nom sur le réseau avec lequel l'ordinateur de l'utilisateur peut désigner cette station. La troisième et dernière fois que ce traducteur est appelé, il s'agit de répondre 8, le plus grand nombre de processeurs AMD 64 bits agrégés dans un même ordinateur.

Traducteur d'implantation de l'architecture ODD/SAMURAAIE Le traducteur d'implantation de l'architecture ODD/SAMURAAIE traduit l'information technologique à propos de l'implantation de l'architecture ODD/SAMURAAIE en éléments et relations entre éléments, pour l'abstraction Application. Dans le cas de déploiement considéré, ce traducteur n'est appelé qu'au démarrage de l'implantation elle-même. Il enregistre un élément Blob d'empreinte en mémoire vive, un élément Activité, et deux éléments Flux. Il relie, en lecture et écriture, les éléments Blob de l'empreinte et Activité avec les deux éléments Flux par des relations d'accès.

Traducteur d'application MPI Le traducteur d'application MPI traduit l'information technologique à propos d'une application MPI en éléments et relations entre éléments, pour l'abstraction Application. Cette information, fournie par un connecteur, comporte la ligne de commande, n le nombre de processus, si ces processus intercommuniquent par mémoire partagée, le chemin d'accès et la taille du fichier binaire exécutable, et un couple de listes des chemins d'accès et des tailles des fichiers lus et écrits pour chaque processus.

Le traducteur d'application MPI est appelé afin d'abstraire l'application de l'utilisateur. Ce traducteur ajoute alors, dans l'abstraction Application, les n éléments Processus, les $2 + 2n$ éléments Blob du répertoire des fichiers de l'application, de la mémoire partagée, des n empreintes en mémoire vive, et des n fichiers de résultats, les n éléments Activité, et les $3n + 3$ éléments Flux. Il relie ces éléments par des relations d'accès et d'agrégation.

Traducteur de pseudo-chemins SAMURAAIE Le traducteur de pseudo-chemins SAMURAAIE traduit des séquences de noms d'éléments en relations de liaison entre ces éléments, pour l'abstraction Carte. Dans ces séquences, les noms peuvent être deux ou quatre et sont séparés par des barres obliques de la manière suivante :

- Nom de l'élément de contenant, barre oblique, et nom de l'élément de contenu ;
- Nom de l'élément d'agrégation de contenant, barre oblique, nom de l'élément simple de contenant, barre oblique, nom de l'élément d'agrégation de contenu, barre oblique, et nom de l'élément simple de contenu.

Ce traducteur est appelé afin d'abstraire la présence effective du répertoire des fichiers de l'application dans le répertoire personnel de l'utilisateur sur son ordinateur et afin d'abstraire la présence attendue des fichiers de résultats dans ce même répertoire.

Traducteur de système GNU/Linux Le traducteur de système GNU/Linux traduit l'information technologique à propos d'un système d'exploitation GNU/Linux d'un ordinateur, pour l'abstraction Services. Cette information, fournie par un connecteur, comporte le nom de l'ordinateur que ce système exploite, le nom et le numéro de version de la distribution, et le numéro de version du noyau. Ce traducteur est appelé afin d'abstraire le système d'exploitation de la station de travail à laquelle l'utilisateur a accès.

Le traducteur de système GNU/Linux ajoute donc un élément Système agrégeant des services de mémorisation et d'exécution, services portant sur les éléments agrégés dans l'ordinateur considéré. Les services de mémorisation de type copie portent également sur les mémoires de masse des ordinateurs avec lesquelles l'ordinateur considéré est connecté. Ces mémoires de masse peuvent ne pas être partagées entre ces ordinateurs. En l'occurrence, le système d'exploitation

de la station peut copier des fichiers ou répertoires depuis et vers le répertoire personnel de l'utilisateur sur son ordinateur.

Traducteur de déploiement Le dernier traducteur, le traducteur de déploiement, traduit les liaisons entre requêtes et services fonctionnant correctement en appels aux connecteurs des technologies de ces requêtes et services. Il traduit donc l'action de copie du répertoire des fichiers de l'application, les actions de démarrage des activités de l'application, et celles de copie des fichiers de résultats en autant d'appels aux connecteurs. Afin de déterminer quel connecteur appeler, le traducteur consulte la caractéristique de technologie des éléments.

Connecteurs

Connecteur POSIX et POSIX par SSH Le connecteur POSIX et POSIX par SSH permet d'abstraire des ordinateurs exploités par des systèmes d'exploitation POSIX, et d'abstraire ces systèmes eux-mêmes. En s'attachant à ces systèmes, il permet également de réaliser des actions sur ces ordinateurs. Ce connecteur est donc appelé afin d'abstraire l'ordinateur personnel de l'utilisateur, la station de travail à laquelle il a accès, leur interconnexion, et le système d'exploitation GNU/Linux de la station. Pour ce faire, il appelle les traducteurs d'ordinateurs et de système GNU/Linux. Ce connecteur est également appelé afin de réaliser la copie du répertoire des fichiers de l'application, le démarrage de ses activités, et la copie de ses fichiers de résultats.

Connecteur MPI Le connecteur MPI permet d'abstraire une application MPI. Il est appelé par l'utilisateur, qui lui transmet l'information technologique pour son application. En implantant une forme simple de comportement adaptatif, ce connecteur permet également de fixer le nombre de processus s'exécutant parallèlement. Pour ce faire, il appelle les traducteurs d'ordinateurs et d'application MPI. Si le nombre de processus s'exécutant parallèlement ne lui est effectivement pas précisé, il consulte d'abord le traducteur d'ordinateurs afin de lui demander le plus grand nombre de processeurs fonctionnant correctement et compatibles avec les activités des processus, puis appelle le traducteur d'application MPI afin d'abstraire l'application comportant autant de processus.

6.4 Déploiement sur grappes d'ordinateurs d'applications paramétriques

Dans cette section, l'utilisateur a accès à une grappe d'ordinateurs multi-processeur. Il veut y déployer une application paramétrique dont il dispose. Après avoir détaillé la grappe et l'application, cette section explique comment l'architecture ODD/SAMURAAIE traite ce cas de déploiement.

6.4.1 Une grappe d'ordinateurs et une application paramétrique

La grappe considérée d'ordinateurs est une grappe de stations multi-processeur de travail, exploitée par TORQUE. Elle comporte 128 de ces stations, chacune ayant deux processeurs Intel Xeon 64 bits. Ces stations ont chacune 4 Go de mémoire vive et un disque dur IDE de 50 Go sur lequel se trouvent un répertoire `/tmp` de fichiers temporaires et un répertoire `/usr/lib` de bibliothèques partagées. Une bibliothèque partagée `librdma.so` de communication par interconnexion Infiniband est présente dans ce dernier répertoire. Ces stations sont localement exploitées par des systèmes GNU/Linux, systèmes acceptant par SSH la soumission à distance de commandes

et par SCP le transfert de fichiers. Ces stations sont interconnectées au sein d'un premier sous-réseau par un commutateur Ethernet Gigabit, d'une part, et dans un second sous-réseau par un commutateur Infiniband, d'autre part.

Les stations de la grappe partagent, par NFS, une mémoire de masse de 500 Go, atteignable par l'utilisateur dans le répertoire `/home/user` sur chacune d'elles. Cette mémoire de masse est ainsi exportée par la seule station reliée au réseau extérieur, la station passerelle. En plus des répertoires `/tmp` et `/usr/lib`, un répertoire `/usr/bin` de fichiers exécutables se trouve également sur le disque dur de cette station. Un fichier exécutable `tar` de manipulation d'archives est présent dans ce répertoire.

L'application considérée est une application maître-esclaves, compilée avec la bibliothèque partagée de communication par interconnexion Infiniband. Compilée pour GNU/Linux Intel 64 bits, cette application comporte les deux fichiers suivants :

- Le premier fichier est le programme maître, un binaire exécutable faisant appel à la bibliothèque partagée. Ce programme accepte quatre paramètres : les nombres de limites inférieure et supérieure de l'étude, le nombre de pas de l'étude, et le chemin d'accès au fichier de résultats.
- Le second fichier est le programme esclave, un autre binaire exécutable faisant appel à la bibliothèque partagée. Ce programme accepte un paramètre : l'adresse du processus maître sur le sous-réseau Infiniband.

De même que pour le cas de déploiement précédent, l'utilisateur a ouvert une session de travail sur son ordinateur personnel. Il a un compte sur la grappe distante d'ordinateurs, grappe dont il connaît l'identifiant sur le réseau et qui, contrairement à la station de travail du cas de déploiement précédent, est partagée par plusieurs utilisateurs. Hors de ses sous-réseaux, cette grappe est effectivement atteignable par la station passerelle. Cette passerelle permet aux utilisateurs de soumettre des tâches à TORQUE et de lire et écrire dans leurs répertoires personnels respectifs. L'ordinateur personnel et la passerelle sont interconnectés au sein du même réseau local, TCP/IP sur Ethernet Gigabit. Quatre puis deux stations de cette grappe sont réservées pour l'utilisateur et la première tombe en panne avant que sa réservation ne soit révolue. Les fichiers de l'application se trouvent dans une archive `appli.tar`, dans le répertoire `/home/user` personnel de l'utilisateur sur son ordinateur. Enfin, l'utilisateur connaît les paramètres de l'étude qu'il veut mener mais ne sait pas combien d'esclaves exécuter.

6.4.2 Traitement avec ODD/SAMURAAIE

Cette sous-section présente d'abord, abstraction par abstraction, comment le scénario de déploiement sur une grappe d'ordinateurs d'une application paramétrique est abstrait avec le modèle SAMURAAIE. Elle présente ensuite, acteur par acteur, comment ce scénario est automatisé avec le modèle ODD. Afin d'en améliorer sa lisibilité, elle ne revient cependant pas sur les concepts déjà illustrés dans le cas de déploiement précédent.

Abstraction avec SAMURAAIE

Ressources La grappe de stations de travail est un ensemble de 128 ordinateurs interconnectés. Dans un premier temps, toutes les stations ne sont qu'enregistrées, sauf la passerelle dont l'état est toujours `OK`. Une fois réservées pour l'utilisateur, en revanche, leur état devient `OK` également. Seul l'état des ordinateurs indique leur réservation ou leur panne.

La double interconnexion des stations de la grappe est abstraite par $2 * 128 = 256$ éléments Connecteur. Pour chaque station, des relations d'accès, en écriture seulement, relient tous les éléments Processeur des autres stations et l'élément Mémoire vive de la station considérée avec

deux de ces éléments Connecteur. Pour chaque paire de connecteurs, le premier a les valeurs caractéristiques du sous-réseau Ethernet Gigabit, le second a celles du sous-réseau Infiniband.

La mémoire de masse partagée par toutes les stations de la grappe est abstraite par un élément Mémoire et quatre éléments Connecteur. Les processeurs de la station passerelle accèdent à cette mémoire par les deux premiers connecteurs, les processeurs de toutes les autres stations par les deux derniers. En l'occurrence, les débits de ces connecteurs sont partagés entre toutes les stations et leurs chemins d'accès sont identiques.

Application En exécutant le fichier exécutable de manipulation d'archives avec les paramètres adéquats, le processus d'installation de l'application lit l'archive et écrit les programmes maître et esclave que cette archive contient. Cette étape du processus de déploiement est donc un élément Processus ordinaire.

Le fichier exécutable de manipulation d'archives et la bibliothèque partagée de communication sont des éléments Blob. Le fichier exécutable n'étant présent que dans un seul répertoire, un seul élément Blob lui correspond. La bibliothèque, en revanche, est présente dans le répertoire des bibliothèques partagées de toutes les stations. Ainsi, 128 éléments Blob lui correspondent. Des relations de copie relient toutefois ces éléments afin de former un cycle. Aucun de ces blobs n'est alors plus original qu'un autre.

De même que dans le cas de déploiement précédent, le nombre de processus esclaves résulte d'une forme de comportement adaptatif. En l'occurrence, ce comportement induit de la dynamicité dans l'application. En effet, le nombre d'esclaves est d'abord quatre, avant que les stations soient réservées, puis il augmente jusqu'à $2 * (4 + 2) = 12$ à mesure que les six stations sont réservées. Il redescend à $12 - 2 = 10$ quand la première station réservée tombe en panne, avant de s'annuler quand ces esclaves ont tous terminé.

Le processus maître et chaque processus esclave sont interconnectés. À chaque interconnexion correspondent deux éléments Flux. Le premier, unique pour toutes les interconnexions, permet aux activités des processus esclaves d'écrire dans l'empreinte en mémoire vive du processus maître. De même, les seconds flux permettent à l'activité du processus maître d'écrire dans les empreintes en mémoire vive des processus esclaves.

Carte Les processus de l'application sont associés aux stations de la grappe. Le processus de manipulation de l'archive des programmes de l'application est associé à la station passerelle. L'archive est alors remplacée par une copie, associée au répertoire personnel dans cette station. Les deux programmes sont également associés à ce répertoire. Le processus maître est associé à la station passerelle. Quand des stations sont réservées, des processus esclaves leur sont associés. Le nombre d'activités ainsi associées est égal au nombre de processeurs disponibles dynamiquement : aucune, huit, douze, dix, puis aucune.

Services Les systèmes d'exploitation des stations ordinaires de la grappe sont similaires à celui de la station passerelle. Leur différence avec celui du cas de déploiement précédent est cependant que ces systèmes n'ont l'état `OK` que quand les stations qu'ils exploitent sont réservées, c'est-à-dire `OK` elles-mêmes. La portée des services de copie de fichiers et de répertoires de ces systèmes reste cantonnée aux sous-réseaux dans lesquels ces stations sont interconnectées.

Programme Au flot de requêtes de services correspondent plusieurs éléments Tâche : un pour l'installation de l'application sur la grappe, un pour l'exécution du maître sur la station passerelle, autant d'éléments Tâche pour l'exécution des esclaves que d'esclaves sur les stations ordinaires, et un pour la copie du fichier de résultats.

L'élément Tâche, pour l'installation de l'application sur la grappe, agrège deux éléments Requête. Le premier demande la copie de l'archive des programmes depuis le répertoire personnel de l'utilisateur sur son ordinateur vers son répertoire personnel sur la station passerelle de la grappe. Le second élément Requête demande le démarrage, sur un processeur de la station passerelle, de l'activité de manipulation de cette copie d'archive. Une relation de précédence relie le premier élément Requête au second.

Chaque élément Tâche, pour l'exécution du maître sur la station passerelle et des esclaves sur les stations ordinaires, agrège un seul élément Requête. Cet élément Requête demande le démarrage de l'activité agrégée dans le processus correspondant. Une relation de précédence relie l'élément Tâche, pour l'installation de l'application sur la grappe, à l'élément Tâche considéré. Ces tâches, pour l'exécution du maître et des esclaves, sont donc parallèles.

De même l'élément Tâche, pour la copie du fichier de résultats, agrège un seul élément Requête. Cet élément Requête demande la copie du fichier de résultats, depuis le répertoire personnel de l'utilisateur sur la station passerelle de la grappe, vers son répertoire personnel sur son ordinateur. Des relations de précédence relient les éléments Tâche, pour l'exécution du maître et des esclaves, à l'élément Tâche considéré. Cette tâche, d'abord en attente, ne peut donc être réalisée qu'après que l'application ait elle-même terminé.

Déploiement Le nombre d'associations de services et de requêtes change quand changent le nombre de systèmes qui fonctionnent correctement et le nombre de tâches en attente, c'est-à-dire quand change le nombre de stations réservées ou en panne pour l'utilisateur.

Capteurs, Messages, et Journal Par rapport au cas de déploiement précédent, les changements survenant pendant le traitement du cas considéré concernent une étape supplémentaire, celle de l'ajustement du déploiement aux comportements dynamiques des ressources et de l'application.

Automatisation avec ODD

Gestionnaires

Associateur Le gestionnaire Associateur constate qu'il ne peut pas associer le processus d'installation ni l'application avec l'ordinateur personnel de l'utilisateur. Il décide donc d'ajouter des copies de l'archive des programmes et du fichier de résultats. Une fois des stations ordinaires disponibles, il associe ce processus d'installation et le processus maître à la station passerelle, les processus esclaves aux stations ordinaires. Quant aux programmes et au fichier de résultats original, afin de ne pas en ajouter davantage de copies, il les associe avec le répertoire que toutes les stations ordinaires de la grappe se partagent : le répertoire personnel de l'utilisateur partagé par NFS.

Tous les processus de l'application ne peuvent pas être associés simultanément, faute de ressources suffisantes. Le gestionnaire Associateur constate cependant que ces processus ne communiquent pas tous directement entre eux. Il décide donc d'en associer une partie seulement : le processus qui communique avec tous les autres, c'est-à-dire le processus maître, et une partie des processus qui ne communiquent qu'avec ce processus, des processus esclaves.

Conseiller La première intervention du gestionnaire Conseiller concerne le fichier de résultats, associé avec le répertoire personnel de l'utilisateur sur son ordinateur. Il constate d'abord que le processus qui écrit ce fichier de résultats, le processus maître, n'est qu'enregistré. Il constate ensuite que le processus maître intercommunique avec d'autres processus qui ne sont

également qu'enregistrés, les processus esclaves. Enfin il constate que le processus qui écrit le fichier que lit le processus maître, le processus d'installation, n'est également qu'enregistré. Il les met alors tous en attente. Une fois associés, Conseiller ajoute les requêtes nécessaires.

Traducteurs Dans le cas de déploiement considéré, l'information traduite porte sur l'ordinateur de l'utilisateur, la grappe d'ordinateurs, leur interconnexion, l'implantation de l'architecture ODD/SAMURAAIE, l'application paramétrique, l'application pour TORQUE, la présence d'éléments de l'application sur des éléments des ressources (les « pseudo-chemins » SAMURAAIE), les systèmes GNU/Linux, la commande `tar`, et les commandes pour l'application paramétrique. À chacune de ces technologies correspond donc un traducteur ODD dont certains ont été présentés dans le cas de déploiement précédent.

Traducteur de grappe d'ordinateurs Le traducteur de grappe d'ordinateurs traduit l'information technologique à propos d'une grappe d'ordinateurs en éléments et relations entre éléments, pour l'abstraction Ressources. Par rapport au traducteur d'ordinateurs, ce traducteur accepte de l'information spécifique à une grappe, telle que le nombre d'ordinateurs et le partage de répertoires. Le traducteur de grappe d'ordinateurs est appelé une fois. Il s'agit d'abstraire la grappe de stations de travail à laquelle l'utilisateur a accès.

Traducteur d'application paramétrique Le traducteur d'application paramétrique traduit l'information technologique à propos d'une application paramétrique en éléments et relations entre éléments, pour l'abstraction Application. Cette information, fournie en paramètre par un connecteur, comporte en particulier n le nombre de processus esclaves et les chemins d'accès et la taille des fichiers exécutables et de résultats. Le traducteur d'application paramétrique est appelé afin d'abstraire l'application de l'utilisateur.

Traducteur d'application pour TORQUE Le traducteur d'application pour TORQUE, le gestionnaire de traitement par lot, traduit une partie des abstractions en information technologique, afin que des ressources puissent être réservées avec TORQUE. Ce traducteur est appelé par un connecteur afin de déterminer le nombre et le type de processeurs ainsi qu'une estimation du temps total de réservation. Ce traducteur est appelé autant de fois que des processus sont en attente de ressources.

Traducteur de commande `tar` Le traducteur de commande `tar`, l'outil de manipulation d'archives, traduit une partie des abstractions en information technologique, afin que les actions spécifiques à la commande `tar` puissent être effectuées. Ce traducteur est appelé par un connecteur afin de traduire les requêtes de démarrage de l'activité du processus d'installation de l'application en une liste d'arguments correspondants aux paramètres de la ligne de commande. Ce traducteur est appelé une seule fois.

Traducteur de commande pour applications paramétriques Le traducteur de programme pour applications paramétriques traduit une partie des abstractions en information technologique, afin que les actions spécifiques à la technologie d'applications paramétriques puissent être effectuées. Ce traducteur est appelé par un connecteur afin de traduire les requêtes de démarrage des activités des processus maître et esclaves de l'application en des listes d'arguments correspondants aux paramètres des lignes de commande. Ce traducteur est donc appelé plusieurs fois.

Connecteurs

Connecteur TORQUE Le connecteur TORQUE permet d'abstraire des ordinateurs exploités localement par des systèmes POSIX et globalement par TORQUE le gestionnaire de lots, et d'abstraire ces systèmes eux-mêmes. En s'attachant aux interfaces de ces systèmes, il permet également de réserver ces ordinateurs et de réaliser des actions sur eux. Ce connecteur est donc appelé afin d'abstraire la grappe de stations de travail à laquelle l'utilisateur a accès et les systèmes d'exploitation TORQUE et GNU/Linux des stations. Pour ce faire, ce connecteur appelle les traducteurs de grappe d'ordinateurs, d'application pour TORQUE, et de système GNU/Linux. Ce connecteur est également appelé afin de réaliser la copie de l'archive des programmes de l'application, le démarrage des activités de l'installation et de l'application, et la copie du fichier de résultats.

Connecteur d'application paramétrique Le connecteur d'application paramétrique permet d'abstraire une application paramétrique. Il est appelé par l'utilisateur afin d'abstraire son application. En implantant une forme simple de comportement adaptatif, il permet également de fixer le nombre de processus esclaves s'exécutant parallèlement. Pour ce faire, il appelle le traducteur d'application paramétrique. Si le nombre de processus esclaves s'exécutant parallèlement ne lui est effectivement pas précisé, il le fixe d'abord à quatre, puis appelle le traducteur d'application paramétrique afin d'abstraire l'application. Tant que tous les processus esclaves fonctionnent correctement, il demande un processus esclave supplémentaire. Quand un processus esclave se termine, il retire un éventuel processus esclave supplémentaire toujours en attente et cesse d'en ajouter.

6.5 Déploiement sur grilles d'ordinateurs d'applications en flot de travail

Dans cette section, l'utilisateur a accès à une grille d'ordinateurs multi-processeur. Il veut y déployer une application en flot de travail dont il dispose. Après avoir détaillé cette grille et cette application, cette section explique comment l'architecture ODD/SAMURAAIE traite ce cas de déploiement.

6.5.1 Une grille d'ordinateurs et une application en flot de travail

La grille considérée d'ordinateurs est une grille de stations de travail multi-processeur. Cette grille comporte deux grappes d'ordinateurs, interconnectées au sein d'un sur-réseau à longue distance. La première grappe comporte 64 stations de travail dont une station passerelle, toutes du type de celle impliquée dans le premier cas de déploiement, et interconnectées au sein d'un sous-réseau TCP/IP sur Ethernet Gigabit. La seconde grappe est identique à celle impliquée dans le deuxième cas de déploiement. Ces grappes, dont les processeurs sont différents, comportent également un outil de manipulation d'archives et une suite de compilation de code source. Enfin, un intergiciel permet de soumettre des commandes et de transférer des fichiers sur toutes les ressources de cette grille.

L'application considérée est un flot de travail à base de composants. Les composants, des bibliothèques partagées, sont chargées dans des conteneurs, un fichier exécutable. Cette application comporte les six composants suivants :

– Un moteur d'exécution de flot de travail. Ce moteur charge, connecte, déconnecte, et décharge des composants en fonction d'une description de flot de travail. La description d'un flot de travail est mémorisée dans un fichier dont le chemin d'accès est fourni en paramètre au moteur d'exécution.

– Un gestionnaire de base de données. Ce gestionnaire lit et écrit un fichier dont le chemin d'accès lui est donné en paramètre. Il peut fonctionner normalement ou en tant que mandataire. Pour le mode de fonctionnement en mandataire, un paramètre supplémentaire indiquant l'adresse du gestionnaire réel est nécessaire.
– Trois programmes séquentiels. Ces programmes manipulent l'information mémorisée dans la base de données.
– Un programme parallèle. Ce programme, dont les processus communiquent par passage de messages, termine le flot de travail et manipule l'information mémorisée dans la base de données de même que les composants séquentiels qui le précèdent dans le flot considéré. Ce programme fiat appel à la bibliothèque partagée de communication par interconnexion Infiniband présentée dans le cas de déploiement précédent.

Le flot lui-même a une forme de losange, le dernier composant étant effectivement le composant parallèle et les deux composants précédents s'exécutant parallèlement. Un fichier de base de données est également fourni pour cette application.

De même que pour les deux cas de déploiement précédents, l'utilisateur a ouvert une session de travail sur son ordinateur personnel. Il a un compte sur la grille distante d'ordinateurs, grille dont il connaît l'identifiant sur le réseau et qui, de même que dans le cas de déploiement précédent, est partagée par plusieurs utilisateurs. Hors de ses sous-réseaux, cette grille est effectivement atteignable par les stations passerelles. Ces passerelles permettent aux utilisateurs de soumettre des commandes et de lire et d'écrire dans leurs répertoires personnels respectifs. L'ordinateur personnel et la passerelle de la seconde grappe sont interconnectés au sein du même réseau local, TCP/IP sur Ethernet Gigabit. Enfin, les fichiers du code source de l'application se trouvent dans une archive `appli.tar`, dans le répertoire `/home/user` personnel de l'utilisateur sur son ordinateur.

6.5.2 Traitement avec ODD/SAMURAAIE

Cette sous-section présent d'abord, abstraction par abstraction, comment le scénario de déploiement sur une fédération de grappes ou grille d'ordinateurs d'une application en flot de travail est abstrait avec le modèle SAMURAAIE. Elle présente ensuite, acteur par acteur, comment ce scénario est automatisé avec le modèle ODD. Pour les mêmes raisons que précédemment, elle ne revient cependant pas sur les concepts déjà illustrés.

Abstraction avec SAMURAAIE

Ressources L'interconnexion de l'ordinateur personnel de l'utilisateur et des deux stations passerelles de la grille est abstraite par six éléments Connecteur. $2 * (1 + 2 + 8) = 22$ relations d'accès relient, en écriture seulement, les éléments Processeur et Mémoire vives avec les éléments Connecteur, à la manière de l'interconnexion des ordinateurs d'une grappe. Ces connecteurs ont des valeurs caractéristiques qui reprennent les caractéristiques topologiques du cas de déploiement considéré. En particulier, la station passerelle de la première grappe étant à longue distance de l'ordinateur personnel et de l'autre station passerelle, la valeur de latence des connecteurs entre cette première et ces seconds est plus grande que celle des connecteurs entre ces seconds.

Application L'outil de manipulation d'archives et la suite de compilation sont quatre éléments Blob dont deux pour la première grappe et deux autres pour la seconde. Ces éléments ont des tailles différentes. En effet, cet outil et cette suite sont différents selon l'architecture de jeu d'instructions des processeurs pour lesquels ils sont compilés. Bien que la suite de compilation comporte plusieurs fichiers exécutables, un seul élément Blob lui correspond.

L'archive du code source de l'application, le fichier de flot de travail, et le fichier de la base de données sont d'autres éléments Blob. Après l'étape d'allocation, des éléments Blob de copie sont également ajoutés. Des relations de copie les relient aux originaux. De plus, pour la base de données, cette copie est modifiée par le processus du composant gestionnaire de base de données. Afin de s'assurer que la base originale et sa copie restent identiques, une seconde relation de copie relie l'original avec la copie, formant ainsi un cycle.

Le processus d'installation de l'application est le processus traditionnel d'installation d'un code source sous GNU/Linux : désarchiver, configurer, compiler, et installer. Pour chaque type de processeur attendu, ce processus est donc abstrait par quatre éléments Processus reliés par des relations de précédence. Les blobs d'entrée et de sortie de ces séquences sont l'archive du code source et un des répertoires des programmes installés. Les processus de l'application sont d'autres éléments Processus reliés par des relations de précédence afin d'abstraire le flot de travail.

Les processus de l'application sont interconnectés à deux niveaux par des éléments Flux. Au premier niveau se trouvent les processus du composant parallèle. Au second niveau se trouvent les processus du moteur d'exécution, du gestionnaire de base de données et de son mandataire, des trois composants séquentiels, et d'un des processus du composant parallèle. En particulier, à cause du caractère dynamique de cette application, aucun de ces flux ne se trouve entre des processus reliés eux-mêmes par une relation de précédence.

Carte La présence du processus d'installation de l'application sur la station passerelle de chaque grappe que comporte la grille est abstraite par deux groupes de relations de liaison. La présence des processus de l'application sur ces grappes est également abstraite par des relations de liaison.

Services De même que ceux des stations de la grappe du cas de déploiement précédent, les systèmes d'exploitation des stations de la grille sont d'abord enregistrés avant de fonctionner correctement pour l'utilisateur, c'est-à-dire pendant que s'écoulent les réservations des ressources sur lesquelles ces systèmes portent.

Programme Au flot de requêtes de services correspondent plusieurs éléments Tâche : un pour l'installation de l'application sur la première grappe, un pour l'installation de l'application sur la seconde, un pour l'exécution du gestionnaire de base de données, un pour l'exécution du moteur d'exécution de flot de travail, trois pour l'exécution des composants séquentiels, un pour l'exécution du mandataire de gestionnaire de base de données, un pour l'exécution du composant parallèle, et enfin un pour la recopie de la base de données.

L'élément Tâche d'installation de l'application sur la première grappe, par exemple, agrège de nombreux éléments Requêtes. Il s'agit par exemple de requêtes de copie de l'archive du code source de l'application, et d'exécution de la compilation. Certains de ces éléments sont reliés entre eux par des relations de précédence et d'autres sont parallèles.

Déploiement Dans l'abstraction Déploiement, des relations de liaison relient par exemple l'élément Système portant sur chaque station passerelle avec l'élément Tâche d'installation de l'application sur chaque grappe. De même que dans le cas de déploiement précédent, des relations de liaison n'apparaissent que quand fonctionne correctement la station sur laquelle porte le système considéré.

Capteurs, Messages, et Journal De même que dans le cas de déploiement précédent, les changements survenant pendant le traitement du cas de déploiement considéré concernent les

quatre étapes d'enregistrement, de demande de déploiement, de déploiement, et d'ajustement. En particulier, les changements de l'étape d'ajustement mettent des stations de travail en état de fonctionnement et les processus de l'application en état de fonctionnement puis de terminaison.

Automatisation avec ODD

Gestionnaires

Associateur De même que dans les cas de déploiement précédents, le gestionnaire Associateur constate que les ressources de l'ordinateur personnel de l'utilisateur ne peuvent être allouées aux activités de l'application. Il décide donc d'ajouter des copies des fichiers que lisent ces activités afin de pouvoir allouer d'autres ressources à celles-ci. Il ajoute en effet deux copies de l'archive du code source, une copie de la description du flot de travail, et une copie de la base de données. Les processus d'installation de l'application sur chaque grappe de la grille ont donc chacun leur copie de l'archive du code source.

Le gestionnaire Associateur relie les copies qui ne sont accédées qu'en lecture aux originaux par des relations de copie et, dans le cas où elles sont également modifiées, les originaux aux copies afin que ces éléments soient mis à jour. C'est le cas de la base de données.

En analysant les abstractions Ressources et Application, le gestionnaire Associateur constate que les stations passerelles doivent être allouées afin que le gestionnaire de base de données et son mandataire puissent intercommuniquer. En effet, l'application est divisée en deux parties par l'architecture demandée de jeu d'instructions des processeurs, cependant ces deux parties intercommuniquent. Bien que la seconde partie, c'est-à-dire le mandataire de gestionnaire et le composant parallèle, ne s'exécute pas au début de l'application, Associateur en tient compte.

Exécuteur En analysant les abstractions Programme et Déploiement, le gestionnaire Exécuteur découvre les relations de précédence entre les éléments Tâche associés à des éléments Système. En particulier, il exécute la tâche d'installation de l'application sur la seconde grappe alors même que l'exécution des derniers composants séquentiels a commencé.

Traducteurs Dans le cas de déploiement considéré, l'information traduite porte sur l'ordinateur de l'utilisateur, la grille d'ordinateurs, leur interconnexion, l'implantation de l'architecture ODD/SAMURAAIE, l'application en flot de travail, l'application pour l'intergiciel de grille, la présence d'éléments de l'application sur des éléments des ressources (les « pseudo-chemins » SAMURAAIE), les systèmes GNU/Linux, la commande `tar` de la séquence de compilation, la séquence de commandes de compilation, et les commandes pour l'application en flot de travail. À chacune de ces technologies correspond donc un traducteur ODD dont certains ont été présentés dans le cas de déploiement précédent.

Traducteur de grille d'ordinateurs Le traducteur de grille d'ordinateurs traduit l'information technologique à propos d'une grille d'ordinateurs en éléments et relations entre éléments, pour l'abstraction Ressources. Par rapport au traducteur de grappe d'ordinateurs, ce traducteur accepte de l'information spécifique à une grille, telle que le nombre d'ordinateurs, le nombre de grappes, et la topologie de ceux-ci. Le traducteur de grille d'ordinateurs est appelé une fois. Il s'agit d'abstraire la grille de stations de travail à laquelle l'utilisateur a accès.

Traducteur d'application en flot de travail Le traducteur d'application en flot de travail traduit l'information technologique à propos d'une application en flot de travail en éléments et relations entre éléments, pour l'abstraction Application. Cette information, fournie en

paramètre par un connecteur, comporte en particulier n le nombre de processus, les relations de précédence entre ces processus, et le chemin d'accès et la taille des fichiers exécutables, d'entrées, et de sorties. Le traducteur d'application en flot de travail est appelé afin d'abstraire l'application de l'utilisateur.

Traducteur d'application pour l'intergiciel de grille Le traducteur d'application pour l'intergiciel de grille traduit une partie des abstractions en information technologique, afin que des ressources puissent être réservées avec cet intergiciel. Ce traducteur est appelé afin de déterminer le nombre et le type de processeurs ainsi qu'une estimation du temps total de réservation. Ce traducteur est appelé autant de fois que des processus sont en attente.

Traducteur de la séquence de commandes de compilation Le traducteur de la séquence de commandes de compilation traduit une partie des abstractions en information technologique, afin que les actions spécifiques à la compilation puissent être effectuées. Ce traducteur est appelé afin de traduire les requêtes de démarrage des activités des processus d'installation de l'application en des listes d'arguments correspondants aux paramètres des lignes de commande. Ce traducteur est appelé deux fois.

Traducteur de commandes d'applications en flot de travail Le traducteur de commandes d'applications en flot de travail traduit une partie des abstractions en information technologique, afin que les actions spécifiques à la technologie d'applications en flot de travail puissent être effectuées. Ce traducteur est appelé afin de traduire les requêtes de démarrage des activités des processus de l'application en des listes d'arguments correspondants aux paramètres des lignes de commande. Ce traducteur est donc appelé plusieurs fois.

Connecteurs

Connecteur d'intergiciel de grille Le connecteur d'intergiciel de grille permet d'abstraire des ordinateurs exploités localement par des systèmes POSIX et globalement par cet intergiciel de grille, ainsi que d'abstraire ces systèmes eux-mêmes. En s'attachant aux interfaces de ces systèmes, il permet également de réserver des ordinateurs et de réaliser des actions sur eux. Ce connecteur est donc appelé afin d'abstraire la grille de stations de travail à laquelle l'utilisateur a accès et les systèmes d'exploitation que sont l'intergiciel de grille et GNU/Linux des stations. Pour ce faire, ce connecteur appelle les traducteurs de grille d'ordinateurs, d'application pour intergiciel de grille, et de système GNU/Linux. Ce connecteur est également appelé afin de réaliser les copies et la recopie de blobs ainsi que le démarrage d'activités de l'application.

Connecteur d'application en flot de travail Le connecteur d'application en flot de travail permet d'abstraire une application en flot de travail. Il est appelé par l'utilisateur afin d'abstraire son application. Pour ce faire, il accepte la description du flot de travail et appelle le traducteur d'application en flot de travail. Parce que l'architecture ODD/SAMURAAIE est dynamique, ce connecteur peut également être appelé de manière interactive afin de modifier le flot de travail pendant son déploiement.

6.6 Discussion

Le modèle SAMURAAIE d'abstraction des systèmes informatiques à haute performance propose des concepts simples qui permettent cependant d'abstraire des cas de déploiement complexes. Les réseaux et sous-réseaux d'ordinateurs hétérogènes, la compilation de code source

conditionné en archive, la copie et recopie de fichiers, et le partage de mémoires de masse sont des exemples de cette complexité. Telles que les cas de déploiement présentés les ont illustré, les abstractions de ces cas complexes sont cependant difficiles à expliciter dans leur totalité car elles ne sont pas prévues pour être parcourues par des humains mais bien par les acteurs du modèle ODD.

Le modèle ODD d'automatisation du déploiement d'applications propose un découpage des rôles qui permet à chaque acteur d'avoir un fonctionnement simple, quelle que soit la complexité des cas de déploiement traités. Ce fonctionnement simple pourrait cependant être enrichi. Tel est par exemple le cas des gestionnaires Conseiller et Exécuteur.

Le gestionnaire Conseiller pourrait par exemple ajouter des requêtes afin de tester la présence d'éléments de contenu, ce qui pourrait éviter la mise en attente d'éléments de contenu produisant les premiers. Par exemple, un fichier de résultats peut avoir été associé avec un répertoire et mis en attente alors qu'il est déjà présent dans ce répertoire. Il n'aurait pas encore été abstrait, produit par exemple par une entité active de l'environnement de déploiement. Une fois exécutée par le gestionnaire Exécuteur, ces requêtes de test ont un résultat : l'état `TERMINÉ` ou `CORROMPU`. Dans le premier cas, Conseiller peut changer directement l'état du fichier pour l'état `TERMINÉ` afin de signifier sa présence. En revanche, si ces requêtes sont en échec, il doit décider quels éléments de contenu en amont mettre en attente.

Le gestionnaire Exécuteur pourrait fournir un mécanisme de tolérance aux défaillances. Puisqu'il analyse les éléments d'instance à la portée des requêtes qu'il exécute avant d'exécuter les requêtes suivantes, Exécuteur peut exécuter de nouveau des requêtes en échec. Par exemple, si l'élément Blob de copie d'un répertoire quitte son état d'attente pour être en échec après que Exécuteur ait exécuté la requête de copie correspondante, alors Exécuteur peut exécuter à nouveau cette requête avant d'exécuter les requêtes suivantes dans le flot des requêtes que la tâche agrège. Si le problème persistait, alors il mettrait la tâche elle-même en échec.

6.7 Conclusion

Les trois cas de déploiement traités ci-dessus ont permis de valider l'architecture de déploiement dynamique ODD/SAMURAAIE. Dans les trois cas, cette architecture a en effet traité l'allocation de ressources, l'installation de l'application sur les ressources allouées, et l'exécution de l'application installée sur ces ressources. De plus, ces trois cas ont introduit les propriétés de parallélisme, d'hétérogénéité, et de dynamicité des systèmes informatiques à haute performance, propriétés que l'architecture ODD/SAMURAAIE sait prendre en charge.

La station, la grappe de stations, et la grille de stations sont toutes parallèles. Les interconnexions de la grappe et les grappes de la grille sont hétérogènes. La grappe et la grille sont également dynamiques. De même, les applications parallèles, paramétriques, et en flot de travail sont toutes parallèles ou réparties. L'application en flot de travail est hétérogène. Les applications paramétriques et en flot de travail sont dynamiques.

Chapitre 7

Automatisation du déploiement de simulations numériques SALOME

7.1 Introduction

Le déploiement dynamique sur des infrastructures informatiques parallèles, hétérogènes, et dynamiques pour la plate-forme de simulation numérique SALOME pose problème. SALOME permet de programmer et exécuter des simulations numériques parallèles, hétérogènes, et dynamiques. Cependant son approche de déploiement contraint fortement l'utilisateur et est restreinte à des infrastructures informatiques parallèles mais homogènes et statiques.

Ce chapitre présente la plate-forme de simulation numérique SALOME et l'approche qu'elle met en œuvre afin de déployer ses simulations numériques. Il montre ensuite comment le déploiement de ces simulations peut être délégué à l'architecture de déploiement dynamique ODD/SAMURAAIE en conservant les mêmes contraintes pour l'utilisateur, présente une implantation de cette délégation, et la valide par une expérience de déploiement d'une simulation numérique avec ANGE. Enfin, ce chapitre décrit les évolutions de SALOME qui permettront de libérer l'utilisateur des contraintes et d'exploiter des infrastructures informatiques hétérogènes et dynamiques pour les simulations numériques.

7.2 SALOME et son approche du déploiement

7.2.1 Introduction

SALOME est une plate-forme mono-utilisateur de simulation numérique. Un utilisateur de SALOME peut y réaliser toutes les étapes d'une simulation numérique, de la définition de structures géométriques à la visualisation de résultats, en passant par la programmation de schémas de calcul. Un utilisateur peut répartir l'exécution de la plate-forme SALOME entre des ressources de calcul à haute performance, grâce à l'infrastructure à base de composants de celle-ci, fondée sur l'architecture d'objets distribués CORBA (*Common Object Request Broker Architecture*). Un développeur de programme de simulation numérique peut intégrer celui-ci sous la forme d'un composant SALOME, afin que des utilisateurs puissent l'utiliser dans leurs simulations.

SALOME est une plate-forme de simulation numérique en production à EDF. EDF R&D anime le partenariat entre de nombreux acteurs scientifiques et techniques, partenariat dont SALOME est issue. Des ingénieurs-chercheurs à EDF et ailleurs utilisent SALOME pour leurs simulations numériques. Lorsqu'ils en développent, ces ingénieurs-chercheurs intègrent leurs programmes de simulation numérique dans SALOME. Utilisant la puissance des ressources de calcul

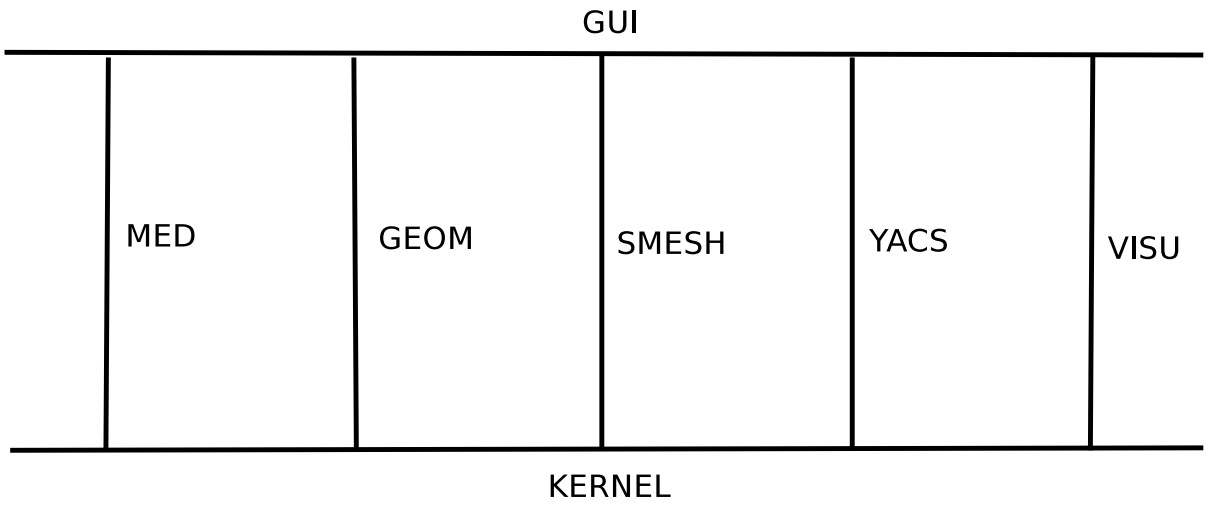


FIGURE 7.1 – Architecture générale de la plate-forme SALOME

à haute performance, la plate-forme de simulation numérique SALOME est indispensable dans l'activité de nombreux ingénieurs-chercheurs EDF.

Après avoir présenté son architecture générale et son modèle de composants, cette section explique comment la plate-forme SALOME est actuellement déployée.

7.2.2 Architecture générale

L'architecture de SALOME est modulaire. Au cœur, SALOME comporte un module noyau (`KERNEL`) auquel peuvent s'ajouter d'autres modules. Le module noyau offre l'infrastructure de la plate-forme, le modèle de composants SALOME, et gère l'exécution des conteneurs de composants SALOME sur les ordinateurs. Les conteneurs permettent de charger et décharger des composants dans l'infrastructure de la plate-forme. Le module de supervision de couplage (*Yet Another Coupling Supervisor*, `YACS`) offre un superviseur. D'autres modules offrent une interface graphique (`GUI`), d'échange de données (`MED`), de modélisation géométrique (`GEOM`), etc. La figure 7.1 page 132 illustre cette architecture.

Comme son nom l'indique, le module d'interface graphique permet à l'utilisateur et aux autres modules de dialoguer dans un paradigme graphique. Cependant, en vertu des bonnes pratiques de conception informatique, dialoguer selon ce paradigme et donc le module d'interface graphique sont optionnels.

7.2.3 Modèle de composants

Introduction

Dans le modèle de composants SALOME [103], les composants sont appelés « services ». Ces services ont des ports qui ont un type : ports de flot de données, de flux de données, ou de contrôle d'exécution. Un flot de données résulte d'une connexion entre un port de flot de données sortantes d'un service et un port de flot de données entrantes d'un autre service. Quand le premier service se termine, des données sont transmises à l'autre service qui démarre. Au contraire, un flux de données est une transmission de données entre des services s'exécutant parallèlement. Un port de flux de données sortantes peut être connecté à plusieurs ports de flux de données entrantes et inversement. Quant aux ports de contrôle d'exécution, ils permettent de définir des précédences entre les exécutions des services. Un contrôle d'exécution, en l'occurrence

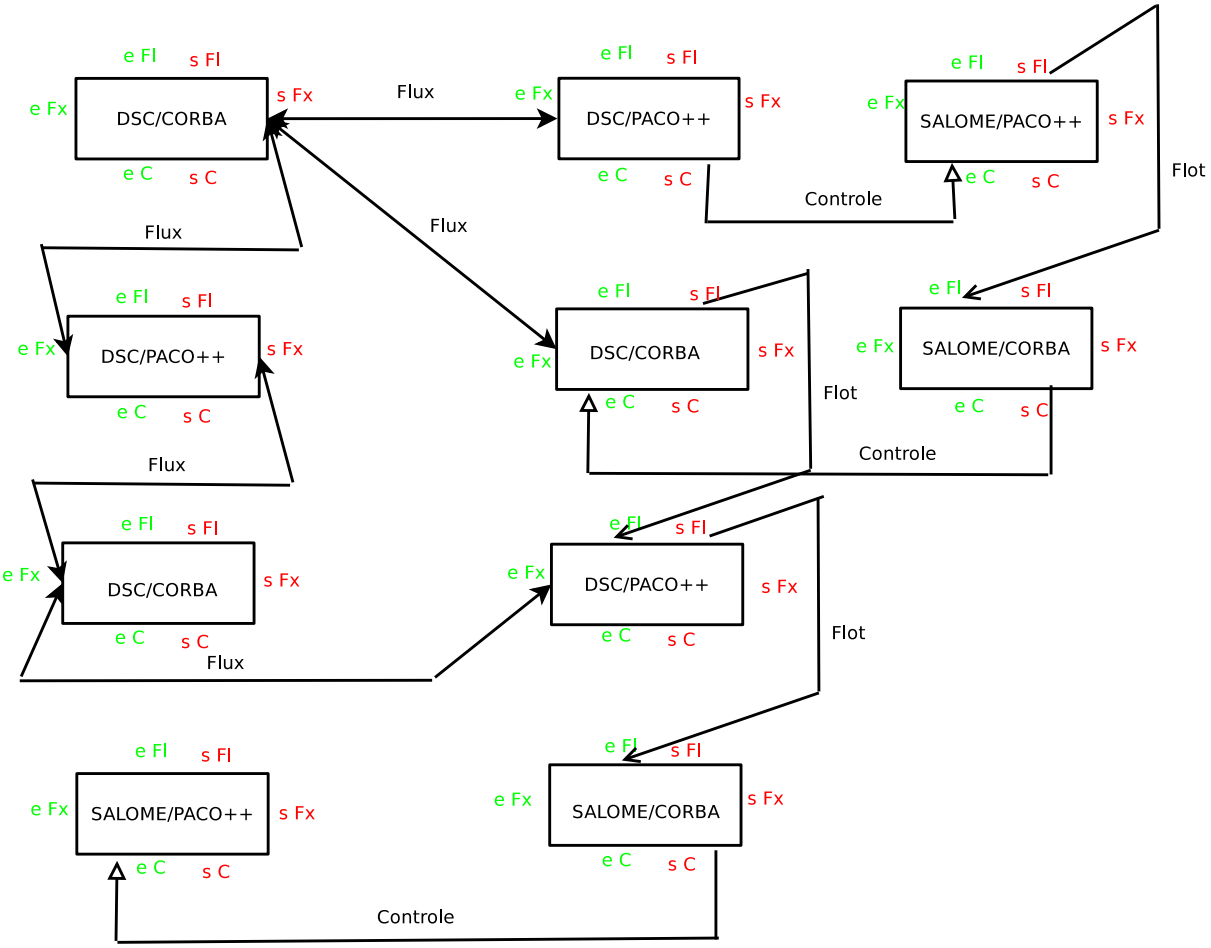


FIGURE 7.2 – Modèle de composants SALOME

le démarrage d'une exécution après la terminaison d'une autre, résulte d'une connexion du port de contrôle d'exécution sortante d'un service et du port de contrôle d'exécution entrante d'un autre service. Parce que les contrôles d'exécution ne font circuler aucune donnée, ces ports et ces connexions sont abstraits. La figure 7.2 page 133 illustre ce modèle de composants.

Le module noyau fournit l'implantation du modèle de composants SALOME. Utilisant les langages de programmation orientée objet C++ et Python, ce modèle est implanté en deux couches, chacune d'elle ayant deux alternatives. La couche de bas niveau comporte des objets CORBA ou des objets PACO++. La seconde couche comporte des objets SALOME ou des composants DSC. Enfin, parmi toutes leurs utilisations possibles et grâce au module de supervision de couplage, les composants SALOME peuvent être intégrés au sein de schémas de calcul. Cette sous-section présente les modèles d'objets PACO++, d'objets SALOME, et de composants DSC, ainsi que le modèle de schémas de calcul.

Objets PaCO++

Le modèle d'objets CORBA parallèles (*Parallel CORBA Object ++*, PaCO++) [104] est un modèle d'objets parallèles fondé sur une extension de CORBA [19]. Cette extension permet d'implanter un objet CORBA en parallèle avec MPI [15], de gérer automatiquement la distribution des données entre les instances parallèles, puis d'utiliser cette implantation dans une application comme s'il s'agissait d'un objet CORBA ordinaire.

Objets SALOME

L'implantation de la plate-forme SALOME se fonde sur l'architecture d'objets répartis CORBA. Un objet SALOME est un objet CORBA respectant la convention suivante :

- Un objet SALOME est un objet CORBA qui implante une interface, appelée `Component`, de gestion de son cycle de vie ;
- Un objet SALOME est un objet CORBA qui offre des ports typés et orientés.

Le principe de conception pour l'implantation des flots de données du modèle de composants SALOME est celui du passage de messages du paradigme de programmation orientée objet. Les ports de flot de données entrantes sont des arguments de méthodes d'objets, les ports de flot de données sortantes, les résultats de méthodes d'objets. Le type des arguments et des résultats de ces méthodes constitue donc le type des ports de flots de données. Les flux de données requièrent cependant une conception plus complexe.

Composants DSC

Le modèle de composants dynamiques (*Dynamic Software Component*, DSC) [105] offre un mécanisme de ports et de connexion de ports dynamiques. De tels composants peuvent en effet, pendant leur exécution, gagner ou perdre des ports. Des connexions entre composants peuvent également être établies pendant leur exécution. Ce mécanisme fournit le principe de conception des flux de données et peut permettre d'autres formes de couplages de services que la transmission de données.

Le modèle de composants dynamiques DSC complète le modèle d'objets SALOME. En effet, un composant dynamique DSC peut respecter les conventions du modèle d'objets SALOME. Ce modèle permet donc au développeur intégrant son programme de simulation numérique dans SALOME de choisir entre les deux sémantiques de transmission de données, flot ou flux, selon les besoins du programme considéré.

Schémas de calcul

Le module de supervision de couplage propose un modèle dans lequel un utilisateur peut définir des nœuds de calcul et en connecter les ports, dans l'espace ou dans le temps. Il peut également définir des conteneurs et leur associer des nœuds de calcul. Ce modèle distingue deux catégories de nœuds, trois catégories de ports, et quatre catégories de connexions.

Les nœuds de calcul peuvent être élémentaires ou composés.

1. Les nœuds élémentaires sont des abstractions de composants SALOME, dont les ports sont définis statiquement. Dans les schémas de calcul, ces nœuds peuvent être associés à des conteneurs, qui abstraient quant à eux des conteneurs de composants SALOME. Quand ils ne sont pas chargés au sein du superviseur, les nœuds sont effectivement exécutés dans des conteneurs de composants SALOME, locaux ou distants grâce à CORBA.
2. Les nœuds composés le sont d'un ou plusieurs nœuds, élémentaires ou bien composés eux-mêmes, connectés logiquement selon une des structures de contrôle suivantes : Bloc, ForLoop, While, ForEach, ou Switch. Les nœuds composés sont des boîtes blanches, c'est-à-dire que tous les nœuds qu'ils comportent peuvent être connectés directement avec d'autres nœuds du schéma de calcul.

Les ports des nœuds sont orientés, typés, et peuvent être de contrôle d'exécution, de données, ou de flux de données. Ces ports permettent au superviseur de couplage, qui interprète les schémas de calcul, d'organiser l'exécution, la circulation des données, et les connexions des nœuds auxquels ils appartiennent.

1. Les ports de contrôle d'exécution permettent au superviseur d'organiser l'exécution des nœuds. Chaque nœud a un port de contrôle d'exécution entrante et un d'exécution sortante.
2. Les nœuds peuvent avoir des ports de données d'entrée, de sortie, ou les deux. Les ports de données permettent au superviseur d'organiser la transmission des données entre les nœuds, respectivement avant, après, ou avant et après l'exécution des nœuds.
3. Enfin, les nœuds peuvent avoir des ports de flux de données entrantes ou de données sortantes. Ces ports permettent au superviseur d'organiser la connexion des nœuds afin que ceux-ci puissent communiquer pendant leur exécution.

Les connexions entre ports des nœuds peuvent être de flot de contrôle, de flot de données, simplement de données, ou de flux de données. Une connexion, de quelque type qu'elle soit, n'est possible qu'entre des ports compatibles, c'est-à-dire de types compatibles et d'orientations complémentaires (depuis les sorties vers les entrées).

1. Les connexions de flot de contrôle connectent des ports de contrôle d'exécution. Pour une connexion de flot de contrôle, une fois terminée l'exécution du nœud auquel appartient le port de contrôle d'exécution sortante, le superviseur démarre l'exécution du nœud auquel appartient le port de contrôle d'exécution entrante.
2. Les connexions de flot de données connectent des ports de données et génèrent des connexions de flot de contrôle entre les nœuds auxquels appartiennent les ports de données impliqués. Ainsi, le superviseur ne fait circuler les données entre les ports que lorsque le permet l'ordre d'exécution fixé par les connexions de flot de contrôle générées.
3. Les connexions simplement de données sont des connexions de flot de données pour lesquelles aucune connexion de flot de contrôle n'est générée.
4. Enfin, les connexions de flux de données connectent des ports de flux de données. Ces connexions peuvent impliquer un port de flux de données sortantes et plusieurs ports de flux de données entrantes, et inversement.

Les étapes par lesquelles passe un nœud pour son exécution s'enchaînent donc ainsi :

1. Le nœud acquiert le contrôle via son port de contrôle d'exécution entrante ;
2. Le nœud acquiert les données d'entrée via ses ports de données d'entrée ;
3. Le nœud s'exécute et communique avec d'autres nœuds via ses ports de flux de données entrantes ou sortantes ;
4. Le nœud fournit les données de sortie via ses ports de données de sortie ;
5. Le nœud rend le contrôle via son port de contrôle d'exécution sortante.

Discussion

Grâce à son modèle de composants, la plate-forme SALOME permet de programmer des simulations numériques parallèles et réparties, hétérogènes, et dynamiques. Pour le parallélisme et la répartition, ces simulations peuvent comporter plusieurs composants SALOME, dont certains peuvent être parallèles eux-mêmes grâce à PACO++, et par ailleurs les nœuds composés peuvent avoir une structure de contrôle parallèle, telle que ForEach. Pour l'hétérogénéité, les composants SALOME étant des objets CORBA, les simulations peuvent donc comporter des implantations de composants dans des paradigmes et des langages de programmation différents. Quant à la dynamicité, les simulations l'obtiennent par le modèle de composants dynamiques DSC, par les connexions de type flot de données, et par les nœuds composés ayant une structure de contrôle dynamique, telle que Switch.

Conclusion

Le modèle de composants de la plate-forme SALOME permet à l'utilisateur de programmer ses simulations numériques sans se préoccuper des alternatives qu'ont choisies les développeurs afin d'intégrer leurs programmes en composants SALOME. Par ailleurs ces simulations peuvent être à la fois parallèles et réparties, hétérogènes, et dynamiques.

7.2.4 Déploiement

Introduction

Le déploiement de simulations numériques SALOME est principalement manuel. L'utilisateur alloue des ordinateurs, y installe SALOME, et SALOME peut alors exécuter les simulations numériques. Cette sous-section présente les étapes d'installation et d'exécution de SALOME, telles qu'elles sont implantées dans la version 4.1.3 du code source ouvert ; puis cette sous-section discute l'approche de déploiement de SALOME dans son ensemble.

Installation

Pour utiliser la plate-forme SALOME afin de réaliser ses simulations numériques, l'utilisateur doit d'abord l'installer. La figure 7.3 page 137 illustre la répartition de modules sur différents ordinateurs. Installer SALOME se déroule selon les trois étapes suivantes :

1. Installer les modules SALOME et leurs dépendances sur des ordinateurs ;
2. Installer les applications SALOME sur les ordinateurs où sont installés les modules et leurs dépendances ;
3. Déclarer auprès de la plate-forme les ressources SALOME préparées dans les deux étapes précédentes.

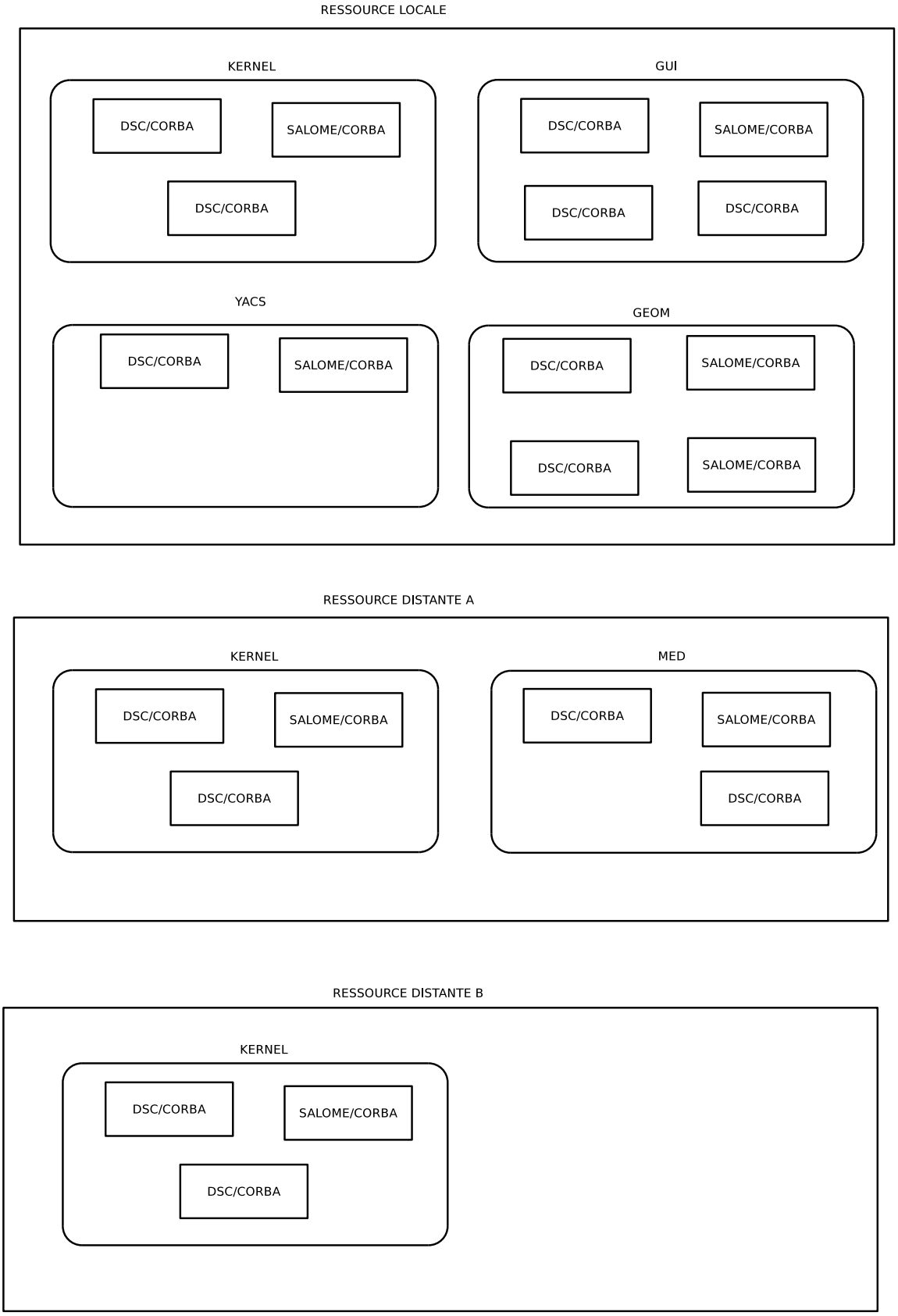


FIGURE 7.3 – Installation de la plate-forme SALOME

Modules Les modules SALOME sont des arborescences de fichiers qui fournissent des parties de l'implantation de la plate-forme elle-même ainsi que des codes intégrés de simulation numérique, leurs dépendances. Un module SALOME comporte des fichiers exécutables, des fichiers d'interfaces et d'implantations de composants SALOME, des fichiers de documentation, des fichiers d'exemples, etc. ainsi qu'un catalogue de module déclarant des meta-données à propos des fichiers précédents dans la perspective de leur utilisation.

Le module noyau (`KERNEL`) fournit les implantations des conteneurs de composants SALOME dans lesquels sont chargés les composants SALOME pendant l'exécution de la plate-forme. Tout module fournissant des implantations de composants a donc besoin du module noyau. Les implantations des conteneurs, qui varient selon les implantations des composants, sont des fichiers exécutables. Il y a deux types d'implantation de conteneurs : les séquentiels et les parallèles. Les séquentiels peuvent contenir des composants SALOME à base d'objets SALOME, les parallèles des composants SALOME à base d'objets PACO++. Ces conteneurs requièrent des développeurs de composants que leurs implantations soient des bibliothèques dynamiques C++ ou bien des modules Python. Le module noyau fournit également des scripts et des squelettes pour la construction et l'exécution d'applications SALOME (voir ci-dessous). Une dépendance du module noyau est OmniORB, une implantation de CORBA comportant notamment un compilateur d'interfaces IDL pour plusieurs langages de programmation ainsi qu'un service de désignation (*naming service*) d'objets CORBA.

Le module de supervision de couplage (`YACS`) fournit quant à lui l'implantation du superviseur. Cette implantation comporte un fichier binaire exécutable et une bibliothèque partagée.

Les fichiers exécutables ou les implantations de composants, que fournissent les modules SALOME, peuvent avoir des dépendances particulières vers des codes externes. L'intégration de codes externes de simulation numérique en composants SALOME repose sur cette possibilité. Installer un module qui aurait des dépendances implique d'installer également ces dépendances.

Un catalogue de module contient toutes les meta-données nécessaires à l'utilisateur afin que celui-ci puisse utiliser les composants SALOME dans ses simulations numériques. En effet, ces meta-données sont par exemple utilisées pour programmer des schémas de calcul. Ces meta-données comportent le modèle de l'implantation des composants, la description de leur interface, etc. Lorsqu'ils contribuent à des modules SALOME en développant des implantations de composants SALOME, les développeurs les déclarent dans les catalogues de modules. Ces catalogues sont pratiquement des fichiers au format XML.

Pour installer les modules SALOME dont il a besoin afin de réaliser ses simulations numériques sur les ordinateurs auxquels il a accès, l'utilisateur a deux solutions. Il peut soit y désarchiver des archives de modules compilés pour le système d'exploitation de ses ordinateurs, soit y compiler le code source de ces modules. Quant aux éventuelles dépendances, la manière de les installer est indépendante de SALOME.

Avant d'utiliser SALOME afin de réaliser ses simulations numériques, l'utilisateur doit encore construire sur chaque ordinateur une vue cohérente des modules qu'il a installés. Cette vue est appelée « application SALOME ».

Applications Les applications SALOME sont des arborescences de fichiers qui offrent une vue cohérente de l'implantation de la plate-forme et des codes intégrés de simulation numérique. Une application SALOME comporte des fichiers exécutables, des implantations de composants SALOME, un script pour son lancement, et des scripts pour la configuration de son environnement. Les fichiers exécutables et les implantations de composants SALOME proviennent des modules SALOME eux-mêmes.

L'utilisateur configure puis génère (au moins) une application SALOME sur chaque ordinateur sur lequel il a installé des modules. Afin de configurer une application, l'utilisateur déclare,

dans un fichier au format XML, la liste des noms et des chemins d'accès aux répertoires d'installation des modules SALOME qu'il a installés. Il peut également faire référence à un script externe de configuration d'éventuelles dépendances vers des codes externes de ces modules. Afin de générer une application, l'utilisateur fait appel à un fichier exécutable que le module noyau lui fournit, et lui passe en paramètre les chemins d'accès au fichier de configuration, d'une part, et au répertoire d'installation de l'application considérée, d'autre part.

En regroupant tous les fichiers que fournissent les modules SALOME dans une même arborescence, les applications SALOME en changent les chemins d'accès. En effet, les chemins d'accès aux répertoires d'exécution diffèrent de ceux des répertoires d'installation. Ceci est rendu possible grâce aux fichiers de configuration des applications, qui déclarent des chemins d'accès aux répertoires d'installation dans lesquels le système d'exploitation cherche les fichiers exécutables pendant l'exécution (variables `PATH`). Ces déclarations font partie de l'environnement d'une application. Par ailleurs, afin d'éviter de copier les fichiers autant de fois que d'applications, l'arborescence générée utilise des liens symboliques.

Avant de l'utiliser afin de réaliser ses simulations numériques, l'utilisateur doit finalement déclarer auprès de la plate-forme elle-même les ordinateurs auxquels il a accès et sur lesquels il a installé des modules puis construit des applications SALOME. Les ordinateurs, les modules, et les applications constituent les « ressources SALOME » qu'il doit déclarer.

Ressources Les ressources SALOME sont les ordinateurs auxquels l'utilisateur a accès et sur lesquels celui-ci a installé la plate-forme, c'est-à-dire des applications SALOME. Autrement dit, sans cette plate-forme installée sur lui, un ordinateur n'est pas une ressource SALOME. Ces ressources sont soit seulement l'ordinateur local de l'utilisateur, soit des ordinateurs en réseau dont fait partie cet ordinateur local. En effet, l'ordinateur sur lequel l'utilisateur démarre SALOME est automatiquement une ressource (»ressource locale ») mais SALOME ne peut utiliser d'autres ordinateurs que si l'utilisateur les lui déclare.

L'utilisateur déclare les ressources SALOME dans un catalogue. Le catalogue de ressources est un fichier au format XML. Afin de construire ce catalogue, l'utilisateur utilise un éditeur. Ce catalogue ne change pas pendant les exécutions de la plate-forme et est personnel à l'utilisateur.

Les meta-données figurant dans le catalogue de ressources SALOME comportent les caractéristiques matérielles des ordinateurs et les caractéristiques de leur système d'exploitation, telles que la capacité de leur mémoire vive, le nom de leur système d'exploitation, et le nom de leur protocole de soumission à distance de commandes (`RSH` ou bien `SSH`) ainsi que l'identifiant de l'utilisateur pour ce protocole. Ces meta-données comportent également, pour chaque ordinateur, le chemin d'accès au répertoire d'installation d'une application SALOME qui y est installée et la liste des noms des modules SALOME que comporte cette application. Par défaut, l'application déclarée pour chaque ordinateur comporte les mêmes modules que l'application de la ressource locale.

Une fois les modules et leurs dépendances installés, les applications construites, et les ressources déclarées, l'utilisateur peut utiliser SALOME afin de réaliser ses simulations numériques.

Exécution

L'exécution d'une simulation numérique SALOME programmée avec un schéma de calcul se déroule en deux étapes :

1. Démarrer la plate-forme SALOME, c'est-à-dire l'application SALOME de la ressource locale. Pour ce faire, l'utilisateur exécute le lanceur de la plate-forme. Ce lanceur démarre alors un intergiciel CORBA et des gestionnaires pour les ressources et pour les conteneurs de composants.

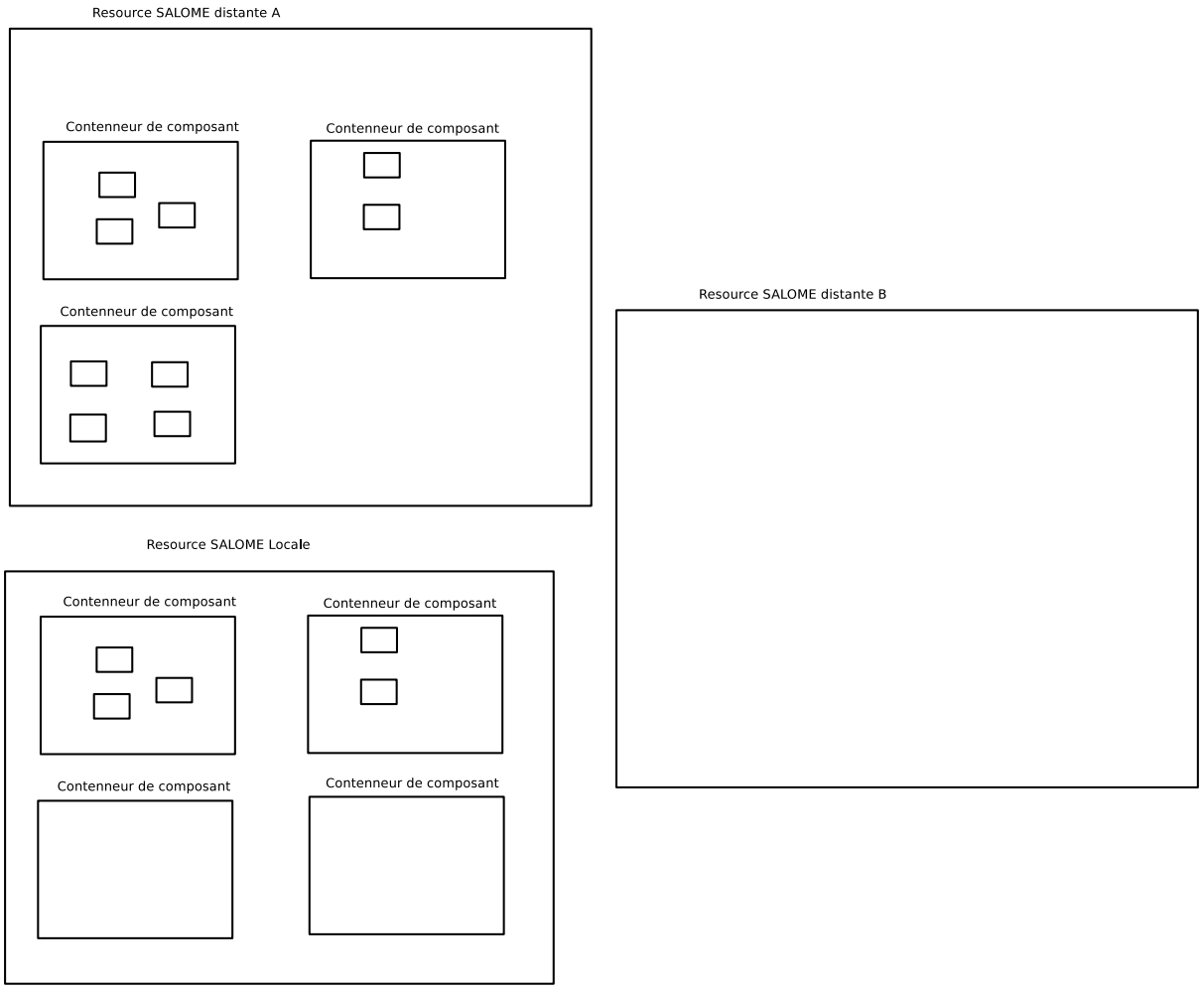


FIGURE 7.4 – Exécution de la plate-forme SALOME

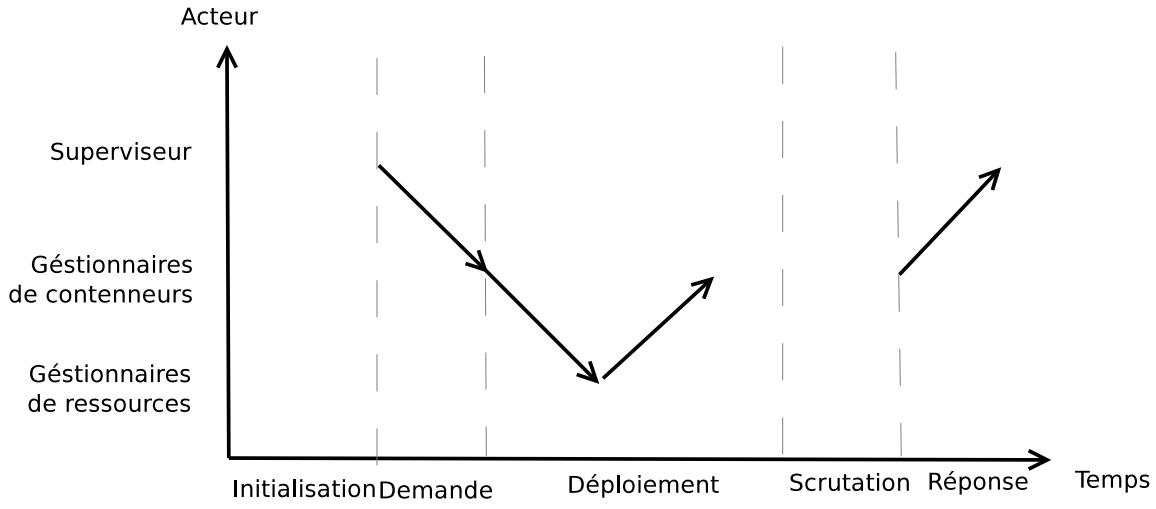


FIGURE 7.5 – Interactions internes de SALOME pour l'exécution de simulations numériques

2. Exécuter la simulation numérique de manière répartie sur les ressources SALOME. Pour ce faire, l'utilisateur démarre le superviseur, dont un paramètre est le chemin d'accès au fichier contenant le schéma, au sein de l'application démarrée sur la ressource locale. Ce superviseur fait alors appel, via CORBA, au gestionnaire de conteneurs de composants SALOME afin d'obtenir des conteneurs. Il charge ensuite les composants SALOME dans ces conteneurs et établit les connexions. La figure 7.4 page 140 illustre la répartition de conteneurs sur différents ordinateurs.

Cette exécution, en deux étapes, implique les trois principaux éléments de la plate-forme SALOME qui suivent (voir la figure 7.5 page 141) :

- Gestionnaire de ressources ;
- Gestionnaire de conteneurs de composants ;
- Superviseur.

Gestionnaire de ressources Le gestionnaire de ressources réalise les trois opérations suivantes :

- Choisir une ressource dans une liste donnée de ressources ;
- Donner la liste des ressources ayant des caractéristiques données ;
- Donner les caractéristiques d'une ressource donnée.

Afin de réaliser ces opérations, le gestionnaire de ressources consulte le catalogue de ressources et les catalogues de modules. Le catalogue de ressources comporte les meta-données des ordinateurs ainsi que les noms des modules qui y sont installés. Les caractéristiques d'une ressource peuvent comporter les caractéristiques de l'ordinateur et des composants que fournissent les modules qui y sont installés. Le gestionnaire utilise donc à la fois les meta-données des ordinateurs et celles des modules qui y sont installés.

Dans son interface, définie conjointement avec celle du gestionnaire de conteneurs de composants dans le fichier `SALOME_ContainerManager.idl`, le gestionnaire de ressources a en effet les trois fonctions suivantes :

`FindFirst` Le paramètre de cette fonction est une liste de noms d'ordinateurs (`possibleComputers`). Cette fonction retourne le premier d'entre eux.

`GetFittingResources` Les paramètres de cette fonction sont des meta-données d'ordinateur (`params`) et une liste de noms de composants (`componentList`). Cette fonction retourne

une liste de noms d'ordinateurs respectant les meta-données données en paramètre et sur lesquelles sont installés les modules fournissant les composants ayant les noms donnés en paramètre.

`GetMachineParameters` Le paramètre de cette fonction est le nom d'un ordinateur. Cette fonction retourne les meta-données de l'ordinateur dont le nom est donné en paramètre.

Le gestionnaire de conteneurs de composants utilise le gestionnaire de ressources.

Gestionnaire de conteneurs de composants Afin de charger et connecter des composants SALOME, il est d'abord nécessaire d'obtenir des conteneurs. Un éléments qui en a particulièrement besoin est le superviseur de couplage. Pour ce faire, le gestionnaire de conteneurs de composants peut réaliser les cinq opérations suivantes :

- Trouver ou démarrer un conteneur de composants séquentiels ;
- Trouver ou démarrer un conteneur de composants parallèles ;
- Démarrer un conteneur de composants séquentiels ou parallèles ;
- Créer un nouveau conteneur de composants ;
- Arrêter tous les conteneurs de composants.

Afin de trouver, démarrer, et créer des conteneurs de composants, le gestionnaire consulte le gestionnaire de ressources et les catalogues des modules installés sur les ordinateurs. En consultant le gestionnaire de ressources, le gestionnaire de conteneurs de composants obtient les meta-données d'un ordinateur sur lequel est installé le module fournissant les composants demandés pour le conteneur demandé ainsi que le chemin d'accès au répertoire d'installation de l'application. Quant aux catalogues de modules, il les consulte afin d'obtenir les meta-données des composants demandés et peut ainsi déterminer l'implantation du conteneur compatible avec l'implantation des composants demandés.

Afin de démarrer ou de créer un nouveau conteneur de composants, une fois toute l'information obtenue du gestionnaire de ressources et des catalogues de modules, le gestionnaire construit puis soumet localement une commande. Cette commande comporte les éléments suivants :

- Le nom de l'implantation du client local du protocole de soumission à distance de commandes de l'ordinateur ;
- L'identifiant de l'utilisateur pour le protocole de soumission à distance de commandes de l'ordinateur ;
- Le nom de l'ordinateur ;
- La commande de positionnement afin que le répertoire d'installation de l'application installée sur l'ordinateur devienne le répertoire courant ;
- La commande d'exécution à distance de l'implantation du conteneur.

La commande d'exécution à distance de l'implantation du conteneur appelle un script (`runRemote.sh`) que fournit l'application installée sur l'ordinateur. Les principaux paramètres de ce script sont l'information nécessaire pour communiquer avec le service de désignation d'objets CORBA, et le nom de l'objet CORBA sous lequel s'enregistrera le conteneur une fois son implantation exécutée.

Une fois la commande complète soumise, le gestionnaire de conteneurs de composants attend la création du conteneur avant de le retourner à l'élément qui le lui avait demandé. Pour ce faire, il scrute l'enregistrement du nom de ce conteneur dans le service de désignation d'objets CORBA. Puis, dès qu'il détecte cet enregistrement, il récupère l'objet CORBA correspondant et le retourne à l'élément qui le lui avait demandé.

Afin d'arrêter les conteneurs de composants, le gestionnaire appelle une fonction des objets CORBA de ceux qu'il avait trouvés, démarrés, ou créés.

Dans son interface, définie dans le fichier `SALOME_ContainerManager.idl`, le gestionnaire de conteneurs de composants a en effet les cinq fonctions suivantes :

`FindOrStartContainer` Les paramètres de cette fonction sont des meta-données d'ordinateur (`params`) et une liste de noms d'ordinateurs. Cette fonction retourne l'objet CORBA d'un conteneur séquentiel.

`FindOrStartParallelContainer` Les paramètres de cette fonction sont les mêmes que ceux de la fonction précédente. En revanche, l'objet CORBA que cette fonction retourne est celui d'un conteneur parallèle.

`StartContainer` Les paramètres de cette fonction sont des meta-données d'ordinateur (`params`), le nom d'une politique d'allocation de ressources (`policy`), et une liste de noms de composants. Cette fonction retourne l'objet CORBA d'un conteneur, séquentiel ou bien parallèle.

`GiveContainer` Les paramètres de cette fonction sont les mêmes que ceux de la fonction précédente. Cette fonction retourne également l'objet CORBA d'un conteneur, séquentiel ou bien parallèle.

`ShutdownContainers` Cette fonction n'a pas de paramètre et ne retourne rien.

Les gestionnaires de ressources et de conteneurs de composants se partagent la gestion du déploiement. Par sa bibliothèque partagée, le superviseur fait partie des éléments qui les utilisent.

Superviseur Afin de l'exécuter, le superviseur interprète un schéma. Il y trouve à la fois les définitions de nœuds de calcul et de conteneurs abstraits, des associations entre des nœuds et des conteneurs abstraits, et des connexions entre nœuds. Il détermine ainsi ce qu'il doit réaliser et ce dont il a besoin pour ce faire :

- Il doit réaliser la répartition des composants parmi les conteneurs, et le flot de contrôle ainsi que les flux de données entre les nœuds ;
- Pour ce faire, il a besoin des composants et de conteneurs dans lesquels charger ces composants.

Pendant l'exécution proprement dite d'un schéma, le superviseur exécute les nœuds de calcul selon leurs connexions de flot de contrôle ou de flot de données, et établit les connexions entre eux selon les flux de données. Afin d'exécuter les nœuds, il demande des conteneurs de composants au gestionnaire de ceux-ci avant d'y charger ces nœuds. En effet, le superviseur fait coïncider les conteneurs abstraits avec des conteneurs de composants.

En faisant appel au gestionnaire de conteneurs de composants, le superviseur le mandate afin d'obtenir des conteneurs respectant deux règles à la fois. La première règle est que les conteneurs doivent être compatibles avec les composants dont le superviseur prévoit le chargement. La seconde règle est que les implantations des conteneurs doivent être exécutées au sein d'applications desquelles font partie les modules qui fournissent ces composants. Par ailleurs, lorsque la définition d'un conteneur abstrait comporte un nom d'ordinateur, l'allocation d'ordinateur pour le conteneur de composants qui coïncide avec lui est forcée.

Discussion

Avant d'utiliser la plate-forme SALOME afin de réaliser ses simulations numériques, l'utilisateur doit construire ses ressources SALOME. Construire ses ressources consiste à allouer des ordinateurs auxquels il a accès, à y installer SALOME, et à déclarer ses ressources SALOME auprès de cette plate-forme. Les applications SALOME ne sont pas les simulations numériques elles-mêmes, elles sont seulement des vues cohérentes de modules SALOME construites sur les ordinateurs alloués. Installer SALOME consiste donc à décider de la répartition des modules SALOME sur les ordinateurs alloués, à les y installer avec leurs dépendances selon cette répartition, et à y construire les applications SALOME. Parce que les applications SALOME ne comportent pas toujours les mêmes modules SALOME, et parce que les répertoires d'installation

des modules SALOME n'ont pas toujours les mêmes chemins d'accès, l'utilisateur peut devoir construire autant d'applications SALOME que d'ordinateurs alloués.

Construire ses ressources SALOME est donc délicat mais nécessaire car l'allocation des ordinateurs par SALOME, qui est répartie entre le gestionnaire de ressources SALOME, le gestionnaire de conteneurs de composants SALOME, et le superviseur, est restreinte aux ressources SALOME. Cette allocation a d'autres restrictions. Par exemple, dans la pile d'exécution ressources, conteneurs, composants, et nœuds, décrite dans la figure ci-dessus, les communications ne sont décrites qu'au niveau des nœuds de calcul. L'allocation de ressources, de conteneurs, et de composants ne peut donc pas en tenir compte.

Tous les niveaux d'exécution se préoccupent des ressources. Afin de trouver, démarrer, ou donner des conteneurs de composants, leur gestionnaire scrute leurs noms dans le service de désignation d'objets CORBA. Ces noms comportent les noms des ordinateurs sur lesquels sont exécutées leurs implantations. Le gestionnaire a donc besoin de connaître les noms des ordinateurs non seulement pour y exécuter à distance les implantations des conteneurs, mais également pour les identifier. Par ailleurs, en plus des noms des composants qu'il veut charger, le superviseur spécifie dans ses demandes de conteneurs les caractéristiques des ordinateurs sur lesquels il veut obtenir ces conteneurs.

Enfin, SALOME traite l'allocation de ressources et l'exécution des applications au niveau des conteneurs de composants et non pas au niveau des composants. Les conteneurs sont pourtant systématiquement alloués en fonction des composants.

Conclusion

En résumé, avant de réaliser des simulations numériques faisant appel à des composants SALOME, l'utilisateur doit d'abord allouer des ordinateurs susceptibles d'exécuter des conteneurs de composants SALOME. Puis il doit y installer des modules SALOME avec leurs dépendances, y construire des applications SALOME, et enfin déclarer ces ressources SALOME auprès de la plate-forme elle-même. Une fois une simulation numérique démarrée, le superviseur fait appel au gestionnaire de conteneurs afin d'obtenir des conteneurs dans lesquels charger et connecter les composants. Le gestionnaire de conteneurs fait alors appel au gestionnaire de ressources afin d'obtenir des noms d'ordinateurs sur lesquels exécuter les conteneurs. L'exécution de conteneurs se fait au sein d'applications, et le chargement de composants implique que les modules qui fournissent les composants fassent partie des applications.

7.2.5 Conclusion

La plate-forme de simulation numérique SALOME est indispensable dans l'activité de nombreux ingénieurs-chercheurs EDF. Cependant, le modèle de déploiement de cette plate-forme contraint à la fois ses utilisateurs, sa performance d'exécution, et sa portabilité.

Le déploiement de la plate-forme SALOME sur les ressources de calcul à haute performance contraint l'utilisateur, et celui-ci n'est pas en mesure d'accepter cette contrainte pour des architectures de ressources dynamiques et hétérogènes. En effet, afin de réaliser ses simulations numériques, l'utilisateur doit allouer les ordinateurs susceptibles d'exécuter des conteneurs de composants SALOME, puis y installer les modules avec leurs dépendances et y construire les applications SALOME, enfin déclarer ces ressources auprès de la plate-forme. L'utilisateur accepte cette contrainte pour un ordinateur local, quelques ordinateurs en réseau, voire une grappe d'ordinateurs utilisant un service de mémorisation attaché au réseau ; en revanche, il n'est pas prêt à l'accepter pour une fédération de grappes ou une grille d'ordinateurs, à cause de l'hétérogénéité et de la dynamicité qu'induisent de telles architectures.

Le déploiement de la plate-forme SALOME sur les ressources de calcul à haute performance contraint également la plate-forme SALOME elle-même, et celle-ci n'est pas en mesure d'exploiter au mieux les performances de calcul offertes par les ressources. En effet, afin d'interpréter des schémas, le superviseur du module `YACS` demande des conteneurs de composants SALOME au gestionnaire de conteneurs et, à son tour, celui-ci demande des ressources au gestionnaire de ressources. Le gestionnaire de ressources en choisit non pas seulement sur des critères de performance, mais également sur des critères d'installation des modules et des applications SALOME. Ce gestionnaire n'est pas en mesure de choisir les ressources les plus performantes si l'utilisateur n'y a pas installé les modules avec leurs dépendances et construit les applications SALOME.

La gestion du déploiement des schémas de calcul sur des ressources de calcul à haute performance, qui dépend fortement des architectures et des systèmes d'exploitation de ces ressources, incombe à la fois à l'utilisateur et à la plate-forme SALOME, et ceux-ci ne sont pas en mesure de tenir compte des évolutions des ressources de calcul à haute performance. En effet, la description des ressources, embarquée dans un catalogue de ressources SALOME, n'est pas indépendante de SALOME parce que les listes de modules et d'applications installés sur les ressources en font partie. Cela complique l'utilisation de systèmes d'information disponibles pour les ressources. La manipulation des ressources est également embarquée dans SALOME et cela complique l'utilisation d'outils de déploiement également disponibles par ailleurs.

La préoccupation du déploiement, proche de celle de nombreuses autres plates-formes applicatives, pourrait être traitée à part. La plate-forme de simulation numérique SALOME bénéficierait alors d'allocations optimisées, d'installations à la volée, et d'exécutions anticipées, le tout indépendamment des architectures et des systèmes d'exploitation des ressources de calcul, de son point de vue.

7.3 Délégation par SALOME du déploiement

7.3.1 Introduction

Afin de résoudre le problème du déploiement dynamique des simulations numériques SALOME, le premier objectif était que SALOME délègue le déploiement de ses conteneurs de composants sans modifier l'utilisation de SALOME. Les trois étapes suivantes ont permis d'atteindre ce premier objectif :

1. Intégrer l'approche de déploiement de SALOME et l'architecture de déploiement dynamique ODD/SAMURAAIE ;
2. Implanter cette intégration en étendant ANGE pour la technologie SALOME, d'une part, et en modifiant l'implantation de SALOME afin qu'elle utilise ANGE pour déployer des conteneurs de composants, d'autre part ;
3. Valider cette intégration et son implantation en réalisant une simulation numérique SALOME pour laquelle ANGE déploierait effectivement des conteneurs de composants.

7.3.2 Intégration de SALOME et d'ODD/SAMURAAIE

Intégrer l'approche de déploiement de SALOME et l'architecture de déploiement dynamique ODD/SAMURAAIE a consisté à capter les données se rapportant au déploiement dans SALOME, puis à traduire ces données en abstractions SAMURAAIE. La figure 7.6 page 146 illustre cette intégration. Ces deux étapes, la captation et la traduction, reviennent à concevoir des connecteurs et des traducteurs ODD pour la technologie SALOME.

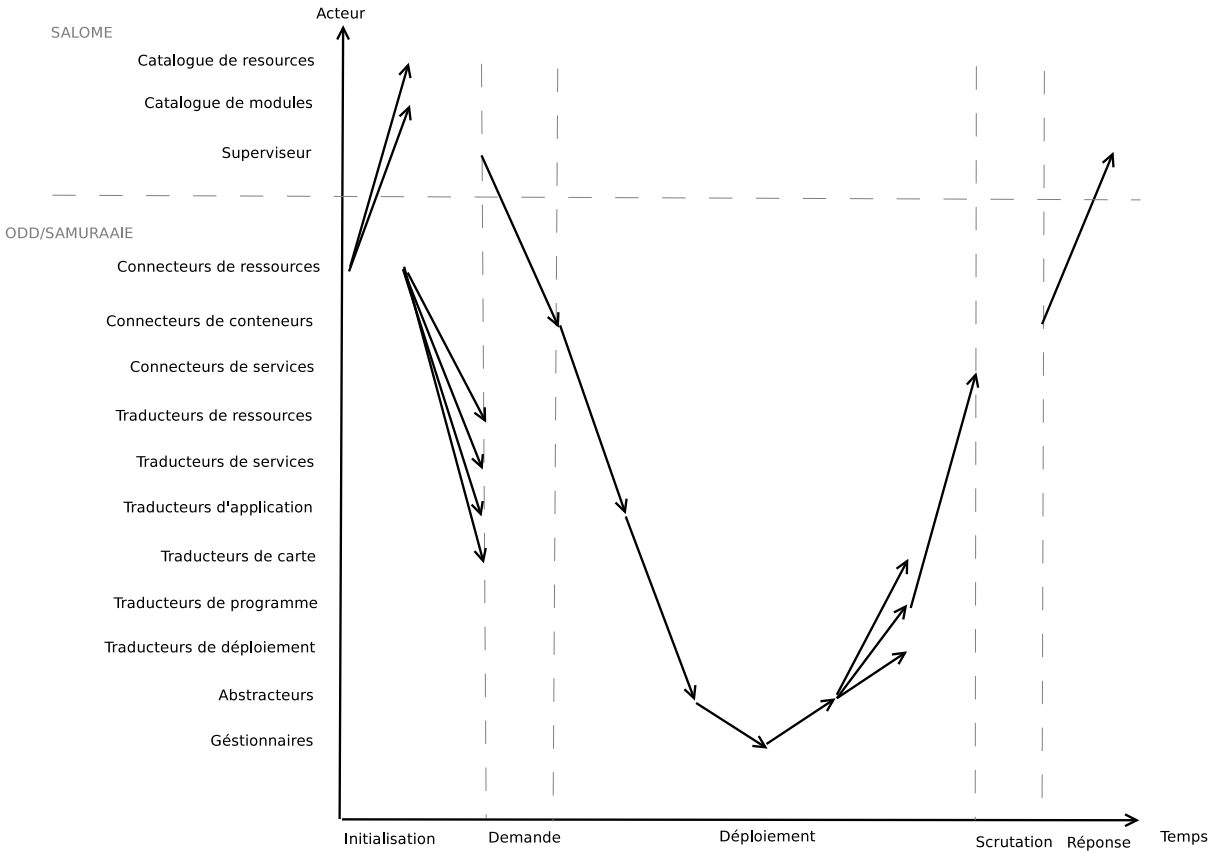


FIGURE 7.6 – Intégration de SALOME et d'ODD/SAMURAAIE

Données SALOME	Abstractions SAMURAAIE
Catalogues de modules	Application
Catalogue de ressources	Ressources, services, application, et carte
Demandes de conteneurs	Application

TABLE 7.1 – Correspondance entre les données de déploiement SALOME et les abstractions SAMURAAIE

Captation des données de déploiement par des connecteurs ODD pour SALOME

Dans l'approche de déploiement de SALOME, les éléments qui allouent des ressources et qui y exécutent des conteneurs de composants sont les gestionnaires de ressources et de conteneurs. Ces gestionnaires manipulent les données de déploiement suivantes :

Catalogues de modules Les catalogues de modules comportent les meta-données des composants, telles que le modèle de leur implantation, leurs interfaces, etc.

Catalogue de ressources Le catalogue de ressources comporte les meta-données des ordinateurs, les chemins d'accès aux répertoires d'installation des applications qui y sont installées, et les noms des modules que ces applications comportent.

Demandes de conteneurs Chaque demande de conteneur de composants comporte des meta-données d'ordinateur, le nom d'une politique d'allocation d'ordinateur, et la liste des noms des composants que doit fournir l'application installée sur l'ordinateur à allouer.

Les acteurs de l'architecture de déploiement dynamique ODD/SAMURAAIE qui captent les données de déploiement spécifiques à des technologies sont les connecteurs. On a donc choisi de remplacer ces deux gestionnaires par deux connecteurs ODD conçus pour la technologie SALOME.

En plus de capter des données de déploiement afin de les transmettre à des traducteurs ODD, les connecteurs ODD peuvent reprendre certaines activités des éléments qu'ils remplacent. Tel est le cas des connecteurs remplaçant les gestionnaires de ressources et de conteneurs de composants de SALOME.

Le connecteur remplaçant le gestionnaire de ressources, appelé connecteur de ressources, ne donne pas de nom d'ordinateur sur lequel exécuter un conteneur comme le faisait pourtant l'élément qu'il remplace. Cette activité revient désormais à l'architecture de déploiement dynamique ODD/SAMURAAIE. N'étant appelé que par le gestionnaire de conteneurs de composants qui est lui-même remplacé, ce changement n'a aucun impact dans SALOME.

En effet, le connecteur remplaçant le gestionnaire de conteneurs de composants, appelé connecteur de conteneurs de composants, n'appelle pas le gestionnaire de ressources comme le faisait l'élément qu'il remplace. Par ailleurs, il n'exécute pas de conteneurs de composants non plus. Cette activité revient désormais à l'architecture de déploiement dynamique ODD/SAMURAAIE. Toutefois, il continue de scruter l'enregistrement de conteneurs dans le service de désignation d'objets CORBA afin de retrouver et de donner les conteneurs aux éléments qui les lui demandent, dont le superviseur.

Une fois captées par les connecteurs de ressources et de conteneurs de composants, les données de déploiement sont transmises à des traducteurs ODD pour la technologie SALOME.

Traduction des données de déploiement par des traducteurs ODD pour SALOME

Les données de déploiement que les connecteurs transmettent aux traducteurs comportent les données suivantes (voir le tableau de correspondance 7.1 page 147) :

Catalogues de modules Les catalogues de modules comportent les meta-données des composants parmi lesquelles le modèle de leur implantation et le nom de leur implantation. Pour des composants donnés, ces meta-données déterminent l'implantation du conteneur dans lequel ces composants pourront être chargés. Elles participent donc indirectement de l'abstraction SAMURAAIE de l'application.

Catalogue de ressources Le catalogue de ressources comporte les meta-données des ordinateurs parmi lesquels figurent leur capacité en mémoire vive, le nom de leur système d'exploitation, le nom de leur protocole de soumission à distance de commandes, et l'identifiant de l'utilisateur pour ce protocole ; les chemins d'accès aux répertoires d'installation d'applications qui sont installées sur ces ordinateurs ; et la liste des noms des modules que comportent ces applications. Conjugué aux catalogues de modules, le catalogue de ressources détermine la liste des implantations des composants installés sur chaque ordinateur ainsi que les chemins d'accès à leur répertoire d'installation. Les meta-données des ordinateurs caractérisent évidemment les ordinateurs eux-mêmes. Ces données participent donc à la fois des abstractions SAMURAAIE des ressources, des services, de l'application, et de la carte (allocation de ressources de mémorisation à l'application).

Demandes de conteneurs de composants Une demande de conteneur de composants comporte des meta-données d'ordinateur restreignant l'allocation, le nom d'une politique d'allocation, et une liste de noms de composants dont l'élément ayant formulé la demande prévoit le chargement. Pouvant contenir jusqu'au nom d'un ordinateur, les meta-données peuvent forcer l'allocation d'un ordinateur. Conjuguées avec les catalogues de modules, ces demandes déterminent l'implantation du conteneur et les implantations des composants qui doivent pouvoir être chargées dans ce conteneur. Ces demandes participent donc de l'abstraction SAMURAAIE de l'application. L'activité d'allocation incombant désormais à l'architecture de déploiement dynamique ODD/SAMURAAIE, le nom d'une politique d'allocation que comportent ces demandes n'est pas considéré.

Les acteurs de l'architecture de déploiement dynamique ODD/SAMURAAIE qui traduisent en abstractions SAMURAAIE des données de déploiement spécifiques à des technologies sont les traducteurs. On a donc conçu les traducteurs ODD suivants pour la technologie SALOME :

Traducteur de ressources Appelé par le connecteur de ressources, le traducteur de ressources traduit en abstraction SAMURAAIE de ressources la description matérielle des ressources SALOME que fournit le catalogue de ressources.

En pratique, il fournit une fonction, appelée `add_host`, dont les paramètres sont les meta-données relatives au matériel d'ordinateur. Cette fonction ajoute dans l'abstraction SAMURAAIE de ressources les mémoires, les connecteurs, les processeurs, et l'ordinateur, d'une part, et les accès internes et externes à l'ordinateur, d'autre part.

Traducteur de services Appelé par le connecteur de ressources, le traducteur de services traduit en abstraction SAMURAAIE de services les meta-données des ressources SALOME ayant trait aux systèmes, telles que les noms de système d'exploitation et les protocoles de soumission à distance de commandes.

En pratique, il fournit une fonction, appelée `add_system`, dont les paramètres sont les meta-données relatives au système d'ordinateur, et le nom d'un ordinateur sur lequel porte le système. Cette fonction ajoute dans l'abstraction SAMURAAIE de services les services et le système de l'ordinateur, d'une part, et les portées vers les mémoires, les connecteurs, les processeurs, et l'ordinateur, d'autre part.

Traducteur d'application Appelé par les connecteurs de ressources et de conteneurs de composants, le traducteur d'application traduit en abstraction SAMURAAIE d'application la

description des composants figurant dans les catalogues de modules, ainsi que les demandes de conteneurs.

En pratique, il fournit deux fonctions, appelées `add_module` et `add_container`, dont les paramètres sont respectivement les meta-données de modules et les paramètres de demandes de conteneurs. La fonction `add_module` ajoute dans l'abstraction SAMURAAIE d'application le blob correspondant au module. En effet, un module SALOME étant indivisible et ses dépendances vers des codes externes ne figurant pas dans les catalogues de modules, un module est abstrait par un seul blob SAMURAAIE. La fonction `add_container` ajoute dans l'abstraction SAMURAAIE d'application les blobs, les flux, les activités, et les processus, d'une part, et les accès internes et externes aux processus, d'autre part.

Traducteur de carte Appelé par les connecteurs de ressources et de conteneurs, le traducteur de carte traduit en abstraction SAMURAAIE de carte la description des installations d'applications et de leurs modules sur des ordinateurs, description que fournit le catalogue de ressources. Inversement, il traduit en noms d'objets CORBA de conteneurs les associations entre les processus des conteneurs et les ordinateurs, associations qui figurent dans l'abstraction SAMURAAIE de carte.

En pratique, il fournit deux fonctions, appelées `map_module_on_host` et `get_container_name_in_naming_service`, dont les paramètres sont respectivement des noms de modules et d'ordinateurs associés et le nom d'un conteneur préalablement demandé. La fonction `map_module_on_host` ajoute dans l'abstraction SAMURAAIE de carte l'association entre le module et l'ordinateur. La fonction `get_container_name_in_naming_service` construit le nom du conteneur dans le service de désignation d'objet CORBA. Pour ce faire, elle cherche dans l'abstraction SAMURAAIE de carte une association entre le conteneur et un ordinateur.

Traducteur de programme Le traducteur de programme traduit l'abstraction SAMURAAIE de programme en commandes pour les implantations de conteneurs.

En pratique, il fournit une fonction, appelée `args`, dont le paramètre est la requête SAMURAAIE de création d'un conteneur. Cette fonction construit les arguments de la commande permettant d'exécuter le conteneur. Pour ce faire, elle cherche dans l'abstraction SAMURAAIE d'application le processus du conteneur.

Fonctionnement

Une fois conçus les connecteurs et les traducteurs ODD pour SALOME afin de respectivement capter et traduire les données de déploiement, l'intégration de l'approche de déploiement de SALOME et l'architecture de déploiement dynamique ODD/SAMURAAIE fonctionne comme suit :

1. Le connecteur de ressources transmet les meta-données d'ordinateurs ainsi que les noms de modules et les chemins d'accès aux répertoires d'installation d'applications qui y sont installées aux traducteurs ODD de ressources, de services, d'application, et de carte ;
2. Ces traducteurs traduisent ces données de déploiement en abstractions SAMURAAIE puis les transmettent aux abstracteurs ODD correspondants ;
3. Le superviseur demande des conteneurs au connecteur ODD de conteneurs ;
4. Ce connecteur transmet les demandes de conteneurs au traducteur ODD d'application ;
5. Ce traducteur traduit les demandes de conteneurs, modifiant l'abstraction SAMURAAIE d'application via l'abstracteur correspondant ;

6. Les gestionnaires ODD associent les abstractions SAMURAAIE d'application et de ressources, produisant ainsi l'abstraction SAMURAAIE de carte ; à partir de cette abstraction, ils conseillent les actions à entreprendre, produisant l'abstraction SAMURAAIE de programme ; ils associent cette abstraction et celle de services, produisant l'abstraction SAMURAAIE de déploiement ; enfin ils exécutent cette abstraction ;
7. Le traducteur ODD de programme construit les commandes pour les implantations de conteneurs et les transmet au connecteur ODD de services POSIX ;
8. Le traducteur ODD de déploiement transmet l'exécution du déploiement au connecteur ODD de services POSIX ;
9. Le connecteur ODD soumet ces commandes sur les ressources et ainsi sont exécutées les implantations de conteneurs ;
10. Le connecteur de conteneurs demande au traducteur de carte le nom du conteneur dans le service de désignation d'objets CORBA et scrute l'enregistrement de ce conteneur dans ce service ; puis il retourne l'objet CORBA de ce conteneur.

Une fois conçus les connecteurs et les traducteurs ODD pour SALOME, on a poursuivi l'intégration de l'approche de déploiement de SALOME et de l'architecture de déploiement dynamique ODD/SAMURAAIE en l'implantant.

7.3.3 Implantation de l'intégration de SALOME et d'ODD/SAMURAAIE

On a mis en œuvre la délégation à ANGE par SALOME du déploiement des conteneurs de celle-ci en deux étapes :

1. Étendre l'implantation de l'architecture de déploiement dynamique ODD/SAMURAAIE, ANGE, pour la technologie SALOME, notamment en implantant les connecteurs et les traducteurs ODD pour SALOME conçus ci-dessus ;
2. Modifier l'implantation de SALOME afin qu'elle utilise ANGE à la place de ses gestionnaires de ressources et de conteneurs de composants.

Extension d'ANGE pour la technologie SALOME

Étendre ANGE pour une technologie consiste à implanter des connecteurs et des traducteurs pour cette technologie, puis à les ajouter dans l'assemblage d'ANGE. Pour SALOME, on a donc implanté les deux connecteurs de ressources et de conteneurs de composants, d'une part, et les traducteurs de ressources, de services, d'application, de carte, et de programme, d'autre part, tous conçus ci-dessus. Leur implantation fut directe.

Comme pour ANGE, on a développé les connecteurs et les traducteurs avec le langage de programmation Python. Pour les connecteurs, l'implantation de la technologie CORBA que SALOME utilise, `OmniORB`, a permis de générer leurs interfaces Python à partir des interfaces IDL que SALOME fournit pour les gestionnaires qu'ils remplacent. Quant aux traducteurs, on a spécialisé les traducteurs de base que fournit ANGE en leur ajoutant les fonctions spécifiques pour SALOME. Les connecteurs communiquent avec les traducteurs par YAML-RPC, on a donc également implanté les mandataires de traducteurs auxquels les connecteurs font appel. Les connecteurs, les traducteurs, et les mandataires de traducteurs sont disponibles dans le code source d'ANGE.

Une fois ANGE étendu pour la technologie SALOME, on a modifié l'implantation de SALOME afin qu'elle utilise ANGE.

Modification de SALOME pour l'utilisation d'ANGE

Modifier l'implantation de SALOME afin qu'elle utilise ANGE a consisté à embarquer ANGE dans SALOME et à remplacer les implantations des gestionnaires de ressources et de conteneurs de composants par celles des connecteurs ODD pour SALOME.

Les gestionnaires de ressources et de conteneurs de composants étant développés avec le langage C++, on a dû remplacer la pile d'appel en C++ qui en créaient les instances par une autre pile, en Python. Cette pile comportait seulement deux objets : un objet, appelé lanceur, qui créait effectivement les instances de ces gestionnaires, et un fichier exécutable, appelé serveur de lanceur, qui initialisait l'infrastructure CORBA et créait cet objet lanceur. Le lanceur réimplanté en Python crée les instances des mandataires de traducteurs, puis crée les instances des connecteurs en leur transmettant ces premières afin que ces connecteurs puissent communiquer avec les traducteurs. En plus d'initialiser l'infrastructure CORBA et de créer une instance de ce nouveau lanceur, le serveur de lanceur également réimplanté en Python initialise ANGE.

Plusieurs éléments de SALOME utilisent les gestionnaires de ressources et de conteneurs de composants, le lanceur, et le serveur de lanceur. Pour ce faire, soit il s'agit d'objets CORBA et alors ils les retrouvent par leur nom dans le service de désignation d'objets CORBA, soit il s'agit de fichiers exécutables et alors ils les exécutent directement sur le système. On a donc remplacé les noms de ces derniers dans toutes les requêtes au service de désignation d'objets CORBA et dans tous les appels au système.

Les réimplantations du lanceur et de son serveur ainsi que les modifications des éléments afin qu'ils utilisent les connecteurs et les réimplantations elles-mêmes sont disponibles sous la forme d'une série de correctifs pour le code source de SALOME.

Une fois implantée l'intégration de l'approche de déploiement de SALOME et de l'architecture de déploiement dynamique ODD/SAMURAAIE, on a pu déployer une simulation numérique SALOME avec ANGE.

7.3.4 Réalisation d'une simulation numérique SALOME déployée par ANGE

La dernière étape afin de s'assurer que la plate-forme de simulation numérique SALOME délègue effectivement le déploiement de ses conteneurs au moteur de déploiement ANGE a consisté en l'exécution répartie sur plusieurs ordinateurs d'un schéma de calcul faisant appel à des composants SALOME.

Le schéma de calcul et les ordinateurs retenus sont simples. Le schéma calcule une addition de deux nombres réels. En plus des nœuds de calcul permettant de transmettre les opérandes et le résultat de cette opération, ce schéma fait appel au composant séquentiel `AddComponent` d'un module d'exemple (`COMPONENTS`) afin d'effectuer l'addition. Les deux ordinateurs, local et distant, sont deux stations de travail bi-processeur quadri-cœur AMD 64 bits. Les modules noyau, de supervision de couplage, d'échange de données, et d'exemple sont compilés et installés avec leurs dépendances sur les deux ordinateurs. Les applications SALOME correspondantes sont configurées et générées sur chaque ordinateur. ANGE est installé sur l'ordinateur local. Enfin, les ressources sont ajoutées au catalogue de ressources SALOME.

Encore une fois, dans l'approche actuelle du déploiement de SALOME, toute cette préparation incombe à l'utilisateur. Ceci n'est pas remis en cause dans l'étape d'intégration de SALOME et d'ANGE mais le sera dans l'étape suivante, celle de l'enrichissement de SALOME par ANGE au profit de l'utilisateur.

Dynamiquement, ANGE alloue un ordinateur et exécute le conteneur dans lequel le superviseur charge le composant `AddComponent` de sorte que le schéma de calcul s'exécute avec succès. Cette exécution suit les étapes suivantes :

1. Au lancement de l'application SALOME, le serveur de lanceur initialise l'infrastructure CORBA, démarre ANGE, et crée une instance de lanceur ;
2. Le lanceur crée une instance de connecteur de ressources en lui passant les instances de mandataires de traducteurs de ressources, de services, d'application, et de carte ;
3. Le lanceur crée également une instance de connecteur de conteneurs de composants en lui passant une instance de mandataire du traducteur d'application ;
4. Le connecteur de ressources utilise les mandataires de traducteurs dont il dispose afin de déclarer les ordinateurs, les modules, et les associations entre ordinateurs et modules ;
5. Pendant l'exécution du schéma, le superviseur demande au connecteur de conteneurs un conteneur où charger le composant `AddComponent` ;
6. Le connecteur de conteneurs détermine le module dans lequel se trouve le composant `AddComponent` et utilise le mandataire de traducteur dont il dispose afin de déclarer un nouveau conteneur dans l'état `EXPECTED` ;
7. ANGE réagit à cette demande en démarrant un conteneur sur l'ordinateur distant : le gestionnaire ODD Associateur modifie l'abstraction SAMURAAIE de carte afin que le conteneur demandé soit associé à un ordinateur compatible, le gestionnaire ODD Conseiller ajoute des requêtes à l'abstraction SAMURAAIE de programme afin que le conteneur soit exécuté, Associateur modifie l'abstraction SAMURAAIE de déploiement afin que les requêtes demandées soient associées à des services SAMURAAIE, le traducteur de programme construit puis transmet la commande au connecteur ODD de services, le gestionnaire ODD Exécuteur exécute le déploiement, et le traducteur de déploiement transmet cette exécution au connecteur de services ;
8. Pendant qu'ANGE réagit à sa demande, le connecteur de conteneurs scrute l'enregistrement du conteneur demandé dans le service de désignation d'objets CORBA et lorsqu'il le trouve, il le retourne au superviseur ;
9. Une fois le conteneur obtenu, le superviseur y charge le composant et termine l'exécution du schéma : l'addition est alors effectuée.

7.3.5 Discussion

Pour l'étape de réalisation d'un schéma de calcul, si ce schéma est simple, il ne requiert pas moins le déploiement dynamique d'un conteneur de composants SALOME sur un ordinateur distant. Or, pour l'implantation de l'intégration de SALOME et d'ODD/SAMURAAIE comme pour la plate-forme SALOME elle-même, la difficulté est de déployer un conteneur. En effet, du point de vue du connecteur de conteneurs de composants comme de celui du gestionnaire de conteneurs, les conteneurs sont totalement indépendants. La réalisation d'une simulation numérique plus complexe, impliquant davantage de conteneurs de composant, n'aurait donc rien montré de plus sur le caractère dynamique du déploiement.

Une fois l'intégration de SALOME et d'ODD/SAMURAAIE validée, la contrainte pour l'utilisateur d'allouer des ordinateurs puis d'installer et de construire manuellement ses applications SALOME sur les ordinateurs alloués pourrait être supprimée. En effet, si l'utilisateur a déjà alloué des ordinateurs a déjà installé les applications, ODD/SAMURAAIE et son implantation en tiennent compte. Cependant ODD/SAMURAAIE et son implantation pourraient également prendre en charge cette allocation et cette installation à la place de l'utilisateur.

7.3.6 Conclusion

L'intégration de l'approche de SALOME et de l'architecture ODD/SAMURAAIE ont permis d'atteindre l'objectif de délégation, par SALOME, à ANGE, du déploiement des conteneurs de

composants de celle-ci. Lorsque le superviseur demande un conteneur de composants au connecteur de conteneurs, afin d'y charger un composant auquel fait appel le schéma qu'il exécute, ce connecteur modifie l'abstraction SAMURAAIE d'application. ANGE réagit alors à cet événement en allouant un ordinateur compatible et en exécutant à distance, sur cet ordinateur, le conteneur correspondant.

Une fois cette intégration achevée, des évolutions de SALOME devraient être réalisées afin qu'elle puisse bénéficier de toutes les fonctionnalités de déploiement dynamique qu'offrent l'architecture ODD/SAMURAAIE et ANGE son implantation.

7.4 Évolutions de SALOME pour le déploiement dynamique

7.4.1 Introduction

Afin de résoudre le problème de déploiement dynamique de simulations numériques SALOME, le second objectif était que l'architecture de déploiement dynamique ODD/SAMURAAIE enrichisse la plate-forme de simulation numérique SALOME. Cet enrichissement, qui remet en cause l'approche de déploiement de SALOME en proposant des évolutions de la plate-forme, surviendrait à chaque étape du déploiement : à l'allocation, à l'installation, et à l'exécution.

7.4.2 Allocation optimisée

L'architecture ODD/SAMURAAIE pourrait optimiser l'allocation d'ordinateurs pour les conteneurs de composants SALOME. En effet, les algorithmes d'ordonnancement du gestionnaire ODD Associateur tiennent compte de toute l'information disponible dans les abstractions SAMURAAIE de ressources et d'application afin d'établir l'allocation demandée, c'est-à-dire modifier l'abstraction SAMURAAIE de carte. Ces algorithmes tiennent également compte de l'allocation courante, ce qui permet à l'algorithme du gestionnaire ODD Conseiller d'optimiser à son tour le nombre d'actions à réaliser pour l'allocation demandée.

Dans l'intégration présentée dans la section précédente, l'architecture ODD/SAMURAAIE ne dispose pas de suffisamment d'information, à propos des conteneurs, en provenance de la plate-forme SALOME, afin d'optimiser l'allocation des ordinateurs pour les conteneurs. En effet, si ces conteneurs sont déterminés par les demandes au connecteur de conteneurs, les communications qui vont avoir lieu entre les composants SALOME que ces conteneurs vont charger ne le sont pas. Or, les ordinateurs alloués pour deux conteneurs dont les composants communiquent intensément doivent probablement différer des ordinateurs alloués pour deux conteneurs dont les composants ne communiquent pas. En outre, si les schémas de calcul ne quantifient pas les débits, les latences, ni les technologies de communication attendues entre des nœuds de calcul faisant appel à des composants SALOME, au moins ils décrivent indirectement quels composants communiquent.

De même, l'architecture ODD/SAMURAAIE ne dispose pas non plus de suffisamment d'information, à propos des ordinateurs, en provenance de la plate-forme SALOME, afin d'optimiser leur allocation pour des conteneurs. En effet, si ces ordinateurs sont décrits dans le catalogue de ressources, les connecteurs, les interconnexions, et leurs technologies ne le sont pas. Or, les ordinateurs alloués pour deux conteneurs dont les composants communiquent intensément doivent probablement différer des ordinateurs alloués pour deux conteneurs dont les composants ne communiquent pas. En outre, des systèmes d'information à propos des ordinateurs peuvent décrire quelles sont les topologies et les technologies des réseaux d'ordinateurs.

Afin d'optimiser l'allocation d'ordinateurs pour les conteneurs de composants SALOME, il faudrait donc que le superviseur communique au connecteur de conteneurs la topologie détaillée dans laquelle se trouvera le conteneur demandé, d'une part, et que le connecteur de ressources

soit renforcé par un système d'information qui connaisse la topologie détaillée dans laquelle se trouvent les ordinateurs, d'autre part.

7.4.3 Installation à la volée

L'architecture ODD/SAMURAAIE pourrait installer à la volée les modules et les applications SALOME sur les ordinateurs. En effet, les traducteurs ODD peuvent modifier l'abstraction SAMURAAIE d'application afin que les outils d'installation fassent eux-mêmes partie de l'application à déployer, et qu'ils puissent ainsi fournir tout ce dont l'application a besoin sur les ressources qui lui sont allouées.

Dans l'intégration présentée dans la section précédente, l'architecture ODD/SAMURAAIE ne dispose pas d'un outil d'installation d'applications SALOME capable d'ajouter des modules SALOME, afin d'installer à la volée les modules et les applications SALOME sur les ressources. En effet, si installer des modules SALOME et générer des applications SALOME est possible, un soin particulier doit être porté aux dépendances des modules et l'application doit être régénérée à chaque installation de module. Or, des systèmes de gestion de paquets ne nécessitant aucun privilège existent et un concept d'application SALOME dynamique est possible.

Afin d'installer à la volée les modules et les applications SALOME sur les ordinateurs, il faudrait donc utiliser un gestionnaire de paquets ne nécessitant aucun privilège pour les modules SALOME et que l'application SALOME soit dynamique, c'est-à-dire qu'ajouter des modules à une application déjà générée soit possible.

7.4.4 Exécution anticipée

L'architecture ODD/SAMURAAIE pourrait anticiper l'exécution des conteneurs de composants SALOME sur les ordinateurs. En effet, les abstractions SAMURAAIE d'application ont une dimension temporelle. Cette dimension permet aux traducteurs ODD d'abstraire l'avenir connu de l'application, et donc aux gestionnaires ODD d'anticiper le déploiement.

Dans l'intégration présentée dans la section précédente, l'architecture ODD/SAMURAAIE ne dispose pas d'horizon temporel pour les conteneurs, en provenance de la plate-forme SALOME, afin d'anticiper l'exécution des conteneurs de composants SALOME sur les ordinateurs. En effet, les demandes de déploiement de conteneurs parviennent au connecteur de conteneurs au fur et à mesure de l'exécution du schéma de calcul. Or, par définition, les schémas de calcul décrivent un horizon temporel.

Afin d'anticiper l'exécution des conteneurs de composants SALOME sur les ordinateurs, il faudrait donc que le superviseur communique les demandes ultérieures au connecteur de conteneurs.

7.4.5 Conclusion

L'architecture de déploiement dynamique ODD/SAMURAAIE pourrait enrichir la plate-forme de simulation numérique SALOME pour chaque étape du déploiement. Cet enrichissement nécessite cependant les évolutions suivantes :

- Le superviseur devrait communiquer au connecteur de conteneurs la topologie détaillée des conteneurs et les demandes ultérieures de conteneurs ;
- Le connecteur de ressources SALOME devrait être renforcé par un système d'information qui connaisse la topologie détaillé des ordinateurs ;
- Le module noyau devrait utiliser un gestionnaire de paquets ne nécessitant aucun privilège pour les modules SALOME et leurs dépendances ;
- L'application SALOME devrait elle-même être dynamique.

7.5 Conclusion

La plate-forme de simulation numérique SALOME est indispensable dans l'activité de nombreux ingénieurs-chercheurs EDF. Pourtant, son approche de déploiement contraint à la fois ses utilisateurs, sa performance d'exécution, et sa portabilité. De nombreuses plates-formes applicatives ont ce problème en commun, ce qui a donné lieu à des travaux dédiés au déploiement automatique.

La plate-forme de simulation numérique SALOME peut déléguer le déploiement de ses conteneurs de composants à l'architecture de déploiement dynamique ODD/SAMURAAIE. Cette délégation pourrait l'enrichir. L'intégration de l'approche de déploiement de SALOME et de cette architecture a été conçue et implantée. Cette intégration a permis de réaliser, de manière répartie et dynamique, une simulation numérique impliquant un conteneur de composants. L'architecture ODD/SAMURAAIE peut dès lors enrichir SALOME pour chaque étape du déploiement. Cet enrichissement, dû aux décisions des gestionnaires ODD, requiert cependant que davantage d'information soit disponible afin d'améliorer la qualité des abstractions SAMURAAIE.

Cinquième partie

Conclusion

Chapitre 8

Conclusion

8.1 Abstraire les systèmes informatiques afin d'en automatiser l'utilisation

Ce document a montré comment abstraire les systèmes informatiques à haute performance afin d'automatiser le déploiement d'applications dynamiques. Classifier les technologies de ces systèmes a en effet permis de structurer un modèle d'abstraction sur lequel un modèle d'automatisation du déploiement d'applications a pu être fondé. Une preuve de concept d'une telle architecture a pu être implantée. Enfin cette architecture et son implantation ont pu être validées avec trois cas de déploiement de complexité croissante ainsi qu'avec une étude du déploiement de simulations numériques SALOME.

Le modèle SAMURAAIE est fondé sur des concepts simples mais il permet d'abstraire des systèmes informatiques complexes. Les contraintes architecturales et logistiques sont abstraites selon le concept de correspondance entre contenants et contenus. Les ressources et l'application sont abstraites, dans le temps, selon leurs éléments d'information, de traitement, et de communication. En particulier, leurs éventuelles propriétés de parallélisme, d'hétérogénéité, et de dynamicité peuvent être abstraites. Les services des systèmes d'exploitation ont un type et portent sur des ressources et, de même, les requêtes des tâches pour ces systèmes d'exploitation ont un type et portent sur l'application. Les événements étant en quelque sorte le dual des actions, la manière de les abstraire est similaire. La généralité de SAMURAAIE est telle qu'il est facilement extensible et pourrait probablement être adapté dans d'autres contextes.

Le modèle ODD de déploiement sur demande, à base d'acteurs concurrents, est un mécanisme permanent de prise de décision doublé d'un mécanisme de représentation interne de l'environnement. Dans la logique du déploiement, les gestionnaires Associateur et Conseiller définissent un objectif et planifient les actions afin de l'atteindre. Les gestionnaires Exécuteur et Contrôleur se chargent de réaliser les actions et de traiter les événements. Cette logique de déploiement et celles des technologies sont séparées grâce aux traducteurs et aux connecteurs. En tant que mécanisme de prise de décision, la qualité de l'information et de ses algorithmes détermine la qualité des décisions.

L'architecture ODD/SAMURAAIE proposée dans ce document a permis d'automatiser l'allocation de ressources, l'installation, et l'exécution d'applications parallèles, paramétriques, et en flot de travail sur un ordinateur, une grappe d'ordinateurs, et une grille d'ordinateurs. Elle a également permis d'automatiser le déploiement de simulations numérique SALOME. Implanté en Python, le moteur de déploiement ANGE est séquentiel et centralisé. Dans ANGE, les données sont mémorisées dans une base de données et les algorithmes sont naïfs. Cette implantation n'a donc pas permis de livrer des mesures de performance. En revanche, elle a permis d'implanter des traducteurs et connecteurs, en particulier pour le déploiement automatique des simulations

numériques SALOME.

Le moteur ANGE a permis d'envisager une place possible pour le déploiement automatique dans les systèmes informatiques à haute performance actuels, À la rencontre des préoccupations des administrateurs, des développeurs, et des utilisateurs. Grâce au modèle ODD, ce moteur peut effectivement être à la fois un outil personnel de l'utilisateur et un intergiciel pour des plates-formes applicatives, au-dessus des systèmes d'exploitation traditionnels que comportent ces systèmes. On imagine aisément l'intégration d'un tel outil au sein de passerelles réservées à l'accès aux systèmes informatiques à haute performance, telles que dans les stations frontales d'aujourd'hui ou les portails pour le calcul scientifique de demain.

8.2 Perspectives

Les travaux que ce document a présentés offrent plusieurs perspectives. À court terme, il s'agirait de poursuivre l'effort d'ingénierie afin de parfaire les développements entamés et de transférer ces travaux en production. À moyen terme, il s'agirait de faire évoluer l'architecture ODD/SAMURAAIE afin de supporter les évolutions actuelles du domaine de l'informatique à haute performance. Il s'agirait également d'améliorer la performance de l'ordonnancement du déploiement en intégrant des travaux existants dans différents domaines. À plus long terme pourraient être étudiées les questions de l'amélioration des performances des applications grâce à leur déploiement automatique ainsi que l'opportunité de versions réparties et multi-utilisateur de l'architecture ODD/SAMURAAIE et de son implantation. Enfin, ces travaux pourraient déboucher sur une notion nouvelle de système personnel d'exploitation.

8.2.1 Poursuite de l'effort d'ingénierie et transfert en production

La partie validation de ce document a présenté des traducteurs et des connecteurs pour des applications parallèles, paramétriques, et en flot de travail et pour un ordinateur, une grappe d'ordinateurs, et une grille d'ordinateurs. Elle a également suggéré des évolutions de la plate-forme de simulation numérique SALOME afin de supporter l'allocation optimisée de ressources, l'installation à la volée, et l'exécution anticipée de conteneurs de composants. Ces évolutions devront être implantées dans la plate-forme SALOME afin d'aller plus loin dans la validation du moteur de déploiement ANGE.

Le transfert en production des travaux présentés dans ce document comporterait au moins trois étapes. La première serait de choisir un langage de programmation statique avec lequel porter la concurrence du modèle d'automatisation de déploiement ODD dans une nouvelle implantation d'ANGE. L'étape suivante serait d'optimiser les mécanismes de mémorisation et de manipulation des abstractions. Enfin, la dernière étape serait d'ajouter à cette implantation le support de technologies de ressources, d'applications, et de systèmes d'exploitation mais également d'interfaces utilisateurs textuelles et graphiques nécessaires en production.

8.2.2 Évolutions de l'architecture ODD/SAMURAAIE

Dans le contexte actuel, la communauté de l'informatique à haute performance s'intéresse aux technologies de virtualisation des ressources, d'hybridation des architectures matérielles, d'hybridation des modèles d'applications, ainsi qu'à l'informatique sur demande ou informatique dans les nuages. Les travaux présentés dans ce document n'ont rien élaboré pour ces sujets et des évolutions sont donc déjà attendues pour l'architecture ODD/SAMURAAIE.

En ce qui concerne les technologies de virtualisation des ressources, l'architecture ODD/-SAMURAAIE pourrait directement les supporter de deux manières. Si la virtualisation ne change pas le fonctionnement des ressources du point de vue de l'utilisateur, alors le support

de ces technologies serait similaire à celui de n'importe quelle autre technologie de ressources, voire celui d'une technologie de ressources déjà supportée. Sinon les utilisateurs peuvent interagir avec les mécanismes de virtualisation, par exemple en devant choisir voire créer leurs ressources virtuelles ou en sachant migrer celles-ci, alors elles seraient considérées par l'architecture ODD/SAMURAAIE comme faisant partie de l'application, seules les ressources réelles sur lesquelles elles s'exécutent feraient partie des ressources. Une autre option pourrait être de considérer non plus une seule frontière application-ressources mais plusieurs, impliquant plusieurs niveaux de déploiement.

L'hybridation des architectures matérielles et celle des modèles d'applications sont déjà supportées dans l'architecture ODD/SAMURAAIE. En effet, le niveau de précision du modèle d'abstraction SAMURAAIE permet par exemple de décrire les processeurs graphiques d'un ordinateur ainsi que l'utilisation de ces processeurs par une application. Puisque les services offerts par les systèmes d'exploitation pour ce type de ressources ne sont pas particuliers, ces ressources seront utilisées par le modèle ODD de la même manière que les processeurs ordinaires. De même, la précision de SAMURAAIE et le fonctionnement actuel d'ODD permettent de déployer des applications ayant par exemple recours autant au modèle de communication par mémoire partagée que par passage de messages. Il resterait cependant nécessaire de valider ces hypothèses avec des cas de déploiement complets.

L'informatique sur demande ou l'informatique dans les nuages est un domaine émergeant auquel l'architecture de déploiement dynamique ODD/SAMURAAIE semble particulièrement adéquate. En effet, la délocalisation des infrastructures et la notion de service, deux caractéristiques de ce domaine, impliquent respectivement une séparation des préoccupations des administrateurs d'infrastructures, des développeurs d'applications, et des utilisateurs d'infrastructures et d'applications, ainsi qu'un besoin en allocation, en installation, et en exécution automatiques. Tels sont justement les résultats des travaux présentés dans ce document. Il resterait cependant à valider cette adéquation avec des cas de déploiement complets.

8.2.3 Vers une meilleure performance de l'ordonnancement du déploiement

Plus les abstractions de l'environnement de déploiement sont fournies et les algorithmes d'ordonnancement intelligents, plus l'ordonnancement du déploiement qui en résulte est performant. Or les représentations ordinaires des infrastructures et des applications peuvent être incomplètes voire inexistantes et les algorithmes utilisés dans ces travaux sont naïfs. Une perspective pour l'architecture ODD/SAMURAAIE serait donc d'avoir une démarche d'acquisition de représentations plus proactive, d'une part, et d'intégrer des dispositifs d'ordonnancement plus efficaces, d'autre part.

Pour l'acquisition d'information, de nombreux travaux existent dans les domaines des infrastructures et des applications. L'architecture ODD/SAMURAAIE pourrait par exemple être couplée avec les travaux en matière de suivi et de découverte de ressources, des outils de profilage d'applications, des environnements d'exécution de langages de programmation répartie, et des cadriciels d'applications à base de composants et d'applications adaptables.

Pour l'ordonnancement, de nombreux travaux existent également dans les domaines de l'intelligence artificielle et de la recherche opérationnelle. Le mécanisme de prise de décision, représenté par les gestionnaires ODD, pourrait par exemple être fourni par des algorithmes avec heuristiques, par des meta-heuristiques, par des solveurs de problèmes de satisfaction de contraintes, ou par des systèmes experts capables d'apprentissage. Ce mécanisme de prise de décision pourrait également compenser les éventuels défauts de fiabilité des systèmes informatiques eux-mêmes en évaluant les risques de pannes et en mettant en place une stratégie de tolérance aux défaillances de la part des infrastructures.

8.2.4 Vers une meilleure performance pour les applications déployées automatiquement

Une perspective importante pour ces travaux est l'amélioration des performances des applications déployées automatiquement sur les infrastructures. Le moteur ANGE peut en effet être utilisé non seulement en tant qu'outil de déploiement automatique mais également comme un outil de suivi de tests d'effort autant pour les infrastructures que pour les applications. De plus, en allouant des ressources pour l'installation et pour l'exécution des applications, de nombreuses optimisations sont d'ores et déjà envisageables. Il s'agirait par exemple d'étendre le modèle SAMURAAIE afin que des critères de consommation énergétique soient pris en compte dans l'ordonnancement.

Deux questions se posent également quant aux évolutions possibles de l'architecture ODD/-SAMURAAIE et de son implantation ANGE. La première concerne le passage à l'échelle du mécanisme actuel de prise de décision. Comment remplacer l'approche centralisée actuelle par une approche totalement répartie ? La seconde question est peut-être duale de la première. Tel qu'il est proposé dans ce document, le modèle SAMURAAIE est mono-utilisateur. De même, l'objectif du mécanisme de prise de décision du modèle ODD correspond à l'intérêt d'un seul utilisateur. Pourtant, plusieurs utilisateurs seront vraisemblablement amenés à utiliser simultanément des instances de cette architecture afin de déployer leurs applications sur une même infrastructure. Une conception collective de cette architecture pourrait-elle alors remplacer la conception individuelle actuelle ?

8.2.5 Vers un système personnel d'exploitation

Enfin, ces travaux de recherche en déploiement automatique pourraient déboucher sur une notion de système personnel d'exploitation. En effet, une conséquence de l'automatisation du déploiement d'applications est de masquer aux utilisateurs la complexité de l'utilisation des systèmes informatiques auxquels ils ont accès. La personnalisation des environnements de ces utilisateurs sur ces systèmes deviendrait donc parfaitement naturelle et, dans le même temps, les administrateurs pourraient simplifier les systèmes d'exploitation ordinaires. Cette personnalisation concernerait non seulement la liste des applications qui seraient offertes à chaque utilisateur mais également la stratégie d'ordonnancement de ses activités. En d'autres termes, un second niveau d'exploitation se dessinerait alors, un niveau personnalisé par chaque utilisateur : un système personnel d'exploitation.

Bibliographie

[1] « Définition d'un ordinateur dans FOLDOC ». http ://foldoc.org/computer, mai 1995.

[2] Flynn (M. J.), « Some Computer Organizations and Their Effectiveness », *IEEE Transactions on Computers*, vol. C-21, septembre 1972, p. 948–960.

[3] « Top 500 Supercomputing sites ». http ://www.top500.org/.

[4] Dongarra (J. J.), Moler (C. B.), Bunch (J. R.) et Stewart (G.), *LINPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1979.

[5] Luszczek (P.), Dongarra (J.), Koester (D.) *et al.*, « Introduction to the HPC Challenge Benchmark Suite », dans *Proceedings of SuperComputing 2005*, mars 2005.

[6] Bailey (D.), Barszcz (E.), Barton (J.) *et al.*, « The NAS Parallele Benchmarks ». Rapport technique, NASA, mars 1994.

[7] Feng (W.-C.) et Scogland (T.), « The Green500 List : Year One », dans *5th IEEE Workshop on High-Performance, Power-Aware Computing (in conjunction with the 23rd International Parallel & Distributed Processing Symposium), Rome, Italy*, mai 2009.

[8] Moore (G. E.), « Cramming More Components Onto Integrated Circuits », *Electronics*, vol. 38, avril 1965.

[9] Amdahl (G.), « Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities », dans *AFIPS Conference Proceedings*, p. 483–485, 1967.

[10] Sterling (T.), Savarese (D.), Becker (D. J.) *et al.*, « BEOWULF : A Parallel Workstation for Scientific Computation », dans *Proceedings of the 24th International Conference on Parallel Processing, Oconomowoc, WI*, p. 11–14, 1995.

[11] Foster (I.) et Kesselman (C.), éditeurs, *The Grid : Blueprint for a New Computing Infrastructure*. Morgan-Kaufmann, 1999.

[12] Intel (R), *Intel(R) Threading Building Blocks - Reference Manual*, édition Intel(R) Threading Building Blocks, juillet 2009.

[13] Richardson (H.), « High Performance Fortran : history, overview and current developments », avril 1996.

[14] Barbara Chapman (R. v. d. P. Gabriele Jost), *Using OpenMP - Portable Shared Memory Parallel Programming*, coll. « Scientific and Engineering Computation ». Massachusetts Institute of Technology Press, octobre 2007.

[15] Gropp (W.), Lusk (E.) et Skjellum (A.), *Using MPI : Portable Parallel Programming with the Message-Passing Interface*. Massachusetts Institute of Technology Press, édition 2nd edition, 1999.

[16] Korpela (E.), Werthimer (D.), Anderson (D.) *et al.*, « SETI@home-Massively Distributed Computing for SETI », *Computing in Science and Engineering*, janvier 2001.

[17] Anderson (D. P.), « BOINC : A System for Public-Resource Computing and Storage », dans *5th IEEE/ACM International Workshop on Grid Computing*, novembre 2004.

[18] Amar (A.), Bolze (R.), Bouteiller (A.) *et al.*, « DIET : New Developments and Recent Results », dans Lehner et al. (Eds.), éditeur, *CoreGrid Workshop on Grid Middleware (in conjunction with EuroPar'06), Dresden, Germany*, n° 4375, coll. « LNCS », p. 150–170. Springer, août 2006.

[19] OMG, *Common Object Request Broker Architecture (CORBA) Specification, version 3.1*, janvier 2008.

[20] Henning (M.), « A New Approach to Object-Oriented Middleware », *IEEE Internet Computing*, janvier 2004.

[21] Orchard (D.), Booth (D.), McCabe (F.) *et al.*, « Web Services Architecture ». Rapport technique, W3C Web Services Architecture Working Group, 2004.

[22] van der Aalst (W.), Hofstede (A. T.), Kiepuszewski (B.) et Barros (A.), « Workflow Patterns », *Distributed and Parallel Databases*, vol. 14, n° 3, juillet 2003, p. 5–51.

[23] Jordan (D.), Evdemon (J.), Alves (A.) *et al.*, *Web Services Business Process Execution Language, version 2.0*. OASIS, mai 2007.

[24] APACHE, *Apache ODE, Developer Guide, Architectural Overview*.

[25] Szyperski (C.), *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley / ACM Press, 1998.

[26] Bernholdt (D. E.), Allan (B. A.), Armstrong (R.) *et al.*, « A Component Architecture for High-Performance Scientific Computing », *International Journal of High Performance Computing Applications*, vol. 20, n° 2, 2006, p. 163–202.

[27] OMG, *CORBA Component Model Specification, version 4.0*, avril 2006.

[28] E. Bruneton (J. S. T. Coupaye), *The Fractal Component Model, version 2.0-3*, février 2004.

[29] Beisiegel (M.), Blohm (H.), Booz (D.) *et al.*, *SCA Service Component Architecture - Assembly Model Specification, version 1.0*, mars 2007.

[30] Ribes (A.), *Contribution à la conception d'un modèle de programmation parallèle et distribué et sa mise en œuvre au sein de plates-formes orientées objet et composant*. Thèse de doctorat, IRISA, Université de Rennes I, 2004.

[31] Mattsson (H.), Nilsson (H.) et Wikström (C.), « Mnesia - A Distributed Robust DBMS for Telecommunications Applications », dans *In Practical Applications of Declarative Languages : Proceedings of the PADL'1999 Symposium, G. Gupta, Ed. Number 1551 in LNCS*, p. 152–163. Springer, 1999.

[32] Protić (J.), Tomasević (M.) et Milutinović (V.), *Distributed Shared Memory : Concepts and Systems*. IEEE, août 1997.

[33] Jan (M.), *JuxMem : un service de partage transparent de données pour grilles de calculs fondé sur une approche pair-à-pair*. Thèse de doctorat, Université de Rennes I, IRISA, Rennes, 2006.

[34] Pennington (H.), Carlsson (A.) et Larsson (A.), *D-Bus Specification, version 0.12*.

[35] Brunet (É.), « NewMadeleine : ordonnancement et optimisation de schémas de communication haute performance », *Technique et Science Informatiques*, vol. 27, n° 3-4/2008, 2008, p. 293–316.

[36] Mainwaring (A.) et Culler (D.), *Active Message Applications Programming Interface and Coummunication Subsystem Organization*. University of California at Berkeley, septembre 1996.

[37] Hewitt (C.), Bishop (P.) et Steiger (R.), « A Universal Modular ACTOR Formalism for Artificial Intelligence », dans *IJCAI'73 : Proceedings of the 3rd International Joint Conference on Artificial Intelligence, San Francisco, CA, USA*, p. 235–245. Morgan Kaufmann Publishers Inc., 1973.

[38] Stoica (I.), Morris (R.), Karger (D.) *et al.*, « Chord : A Scalable Peer-to-Peer Lookup Service for Internet Applications », dans *Proceedings of the 2001 Conference on Applications, Technologies, aRchitectures, and Protocols for Computer Communications*, p. 149–160, 2001.

[39] Jha (S.), Kaiser (H.), Merzky (A.) et Weidner (O.), « Grid Interoperability at the Application Level Using SAGA », dans *Accepted for International Grid Interoperabilty and Interoperation Workshop 2007 (IGIIW 2007)*, 2007.

[40] Bovet (D. P.) et Cesati (M.), *Understanding the Linux Kernel*. Oreilly, octobre 2000.

[41] Schmuck (F.) et Haskin (R.), « GPFS : A Shared-Disk File System for Large Computing Clusters », dans *FAST'02 : Proceedings of the 1st USENIX Conference on File and Storage Technologies, Monterey, CA, USA*, p. 19. USENIX Association, 2002.

[42] *Lustre Operations Manual, version 1.8*, mai 2009.

[43] Morin (C.), Gallard (P.), Lottiaux (R.) et Vallée (G.), « Towards an Efficient Single System Image Cluster Operating System », *Future Generation Computer Systems*, vol. 20, n° 2, janvier 2004.

[44] Jeanvoine (E.), Rilling (L.), Morin (C.) et Leprince (D.), « Using Overlay Networks to Build Operating System Services for Large Scale Grids », *Scalable Computing : Practice and Experience*, vol. 8, n° 3, 2007, p. 229–239. Special issue on Practical Aspects of Large-Scale Distributed Computing, selected paper and extended version of ISPDC'06.

[45] Laforenza (D.), « XtreemOS : Towards a Grid-Enabled Linux-Based Operating System », dans *BNCOD*, p. 241–243, 2008.

[46] Foster (I. T.), « Globus Toolkit, version 4 : Software for Service-Oriented Systems », *Journal of Computer Science and Technology*, vol. 21, n° 4, 2006, p. 513–520.

[47] Foster (I.), « What is the Grid ? - a three point checklist », *GRIDtoday*, vol. 1, n° 6, juillet 2002.

[48] Russell (R.), Quinlan (D.) et Yeoh (C.), *Filesystem Hierarchy Standard*, 2004.

[49] Carns (P. H.), Ligon III (W. B.), Ross (R. B.) et Thakur (R.), « PVFS : A Parallel File System for Linux Clusters », dans *Proceedings of the 4th Annual Linux Showcase and Conference, Atlanta, GA, USA*, p. 317–327. USENIX Association, 2000.

[50] Andreozzi (S.), Burke (S.), Ehm (F.) *et al.* « GFD.147 : GLUE Specification, version 2.0 ». GLUE-WG/Open Grid Forum, mars 2009. REC.

[51] Anjomshoaa (A.), Brisard (F.), Drescher (M.) *et al.* « GFD.136 : Job Submission Description Language (JSDL) Specification, version 1.0 ». JSDL-WG/Open Grid Forum, septembre 2008. REC.

[52] Dijkstra (E. W.), *Selected writings on computing : a personal perspective*, chap. On the role of scientific thouht, p. 60–66. Springer-Verlag New York, Inc., 1982.

[53] Fowler (M.). « Inversion of Control, Containers and the Dependency Injection Pattern ». http ://www.martinfowler.com/articles/injection.html, janvier 2004.

[54] Garlan (D.), Monroe (R. T.) et Wile (D.), « Acme : Architectural Description of Component-Based Systems », dans Leavens (G. T.) et Sitaraman (M.), éditeurs, *Foundations of Component-Based Systems*, p. 47–68. Cambridge University Press, 2000.

[55] Allen (R.), *A Formal Approach to Software Architecture*. Thèse de doctorat, Carnegie Mellon, School of Computer Science, janvier 1997. Issued as CMU Technical Report CMU-CS-97-144.

[56] Magee (J.), Dulay (N.) et Kramer (J.), « Structuring Parallel and Distributed Programs », *Software Engineering Journal*, vol. 8, 1993, p. 73–82.

[57] Lacour (S.), Pérez (C.) et Priol (T.), « Generic Application Description Model : Toward Automatic Deployment of Applications on Computational Grids », dans *6th IEEE/ACM International Workshop on Grid Computing (Grid2005), Seattle, WA, USA*. Springer-Verlag, novembre 2005.

[58] Bouziane (H.), *De l'abstraction des modèles de composants logiciels pour la programmation d'applications scientifiques distribuées*. Thèse de doctorat, Université de Rennes I, février 2008.

[59] OMG, *OMG Unified Modeling Language (UML) - Infrastructure, version 2.2*, février 2009.

[60] Wolski (R.), Spring (N. T.) et Hayes (J.), « The Network Weather Service : A Distributed Resource Performance Forecasting Service for Metacomputing », *Future Generation Computer Systems*, vol. 15, n° 5–6, 1999, p. 757–768.

[61] Legrand (I.), Newman (H.), Voicu (R.) *et al.*, « MonALISA : An Agent based, Dynamic Service System to Monitor, Control and Optimize Grid based Applications », dans *Proceedings of CHEP'04*, septembre 2004.

[62] Levon (J.), *Oprofile Internals*, 2003.

[63] Nethercote (N.) et Seward (J.), « Valgrind : A Framework for Heavyweight Dynamic Binary Instrumentation », dans *Proceedings of the 2007 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'07), San Diego, CA, USA*, p. 89–100, New York, NY, USA, 2007. ACM.

[64] Shende (S. S.) et Malony (A. D.), « The Tau Parallel Performance System », *The International Journal of High Performance Computing Applications*, vol. 20, 2006, p. 287–331.

[65] Desnos (A.), Roy (S.) et Vanengue (J.), « ERESI : une plate-forme d'analyse binaire au niveau noyau », dans *Proceedings of SSTIC*, mai 2008.

[66] Thain (D.), Tannenbaum (T.) et Livny (M.), « Condor and the Grid », dans Berman (F.), Fox (G.) et Hey (T.), éditeurs, *Grid Computing : Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., décembre 2002.

[67] Gil (Y.), Ratnakar (V.), Deelman (E.) *et al.*, « Wings for Pegasus : Creating Large-Scale Scientific Applications Using Semantic Representations of Computational Workflows », *Proceedings of the N19th Conference on Innovative Applications of Artificial Intelligence (IAAI'07)*, 2007, p. 1767–1774.

[68] Myers (K. L.), « CPEF : A Continuous Planning and Execution Framework », *AI Magazine*, vol. 20, n° 4, 1999, p. 63–69.

[69] Leduc (J.) et Peaquin (G.), *Kadeploy, version 2*, juillet 2004.

[70] Lange (T.), *Manuel d'utilisation de FAI (Fully Automatic Installation), version 2.4.3 pour FAI 2.6*, août 2004. Traduit par D. Leprince.

[71] OSGi Alliance, *OSGi Alliance Specifications verion 4.2*, 2009 septembre.

[72] Claudel (B.), Huard (G.) et Richard (O.), « TakTuk, Adaptive Deployment of Remote Executions », dans *Proceedings of the 18th ACM International Symposium on High Performance Distributed Computing (HPDC'09), Garching, Germany*, p. 91–100, New York, NY, USA, 2009. ACM.

[73] Aiftimiei (C.), Andreetto (P.), Bertocco (S.) *et al.*, « Design and Implementation of the gLite CREAM Job Management Service ». Rapport technique, INFN, mai 2009.

[74] des Ligneris (B.), Scott (S.), Naughton (T.) et Gorsuch (N.). « Open Source Cluster Application Resources (OSCAR) : Design, Implementation and Interest for the [Computer] Scientific Community », 2003.

[75] Sacerdoti (F. D.), Chandra (S.) et Bhatia (K.), « Grid Systems Deployment & Management Using ROCKS », dans *Proceedings of IEEE International Conference on Cluster Computing*, p. 337–345, 2004.

[76] Burgess (M.) et Ralston (R.), « Distributed Resource Administration Using Cfengine », *Softw., Pract. Exper.*, vol. 27, n° 9, 1997, p. 1083–1101.

[77] Lascu (O.), Brindeyev (A.), Quintero (D. E.) *et al.*, *xCAT 2 Guide for the CSM System Administrator*. IBM, novembre 2008.

[78] Siddiqui (M.), Villazón (A.), Hofer (J.) et Fahringer (T.), « GLARE : A Grid Activity Registration, Deployment and Provisioning Framework », dans *Proceedings of the ACM/IEEE SuperComputing Conference on High Performance Networking and Computing (SC'05), Seattle, WA, USA*, p. 52. IEEE Computer Society, novembre 2005.

[79] Cunin (P.-Y.), Lestideau (V.) et Merle (N.), « ORYA : A Strategy Oriented Deployment Framework », dans Dearle (A.) et Eisenbach (S.), éditeurs, *Component Deployment*, vol. 3798 (coll. *Lecture Notes in Computer Science*), p. 177–180. Springer, novembre 2005.

[80] Joita (L.), Rana (O. F.), Chacín (P.) *et al.*, « Application Deployment Using Catallactic Grid Middleware », dans *Proceedings of the 3rd International Workshop on Middleware for Grid Computing (MGC'05), Grenoble, France*, p. 1–6, New York, NY, USA, 2005. ACM.

[81] Toft (P.) et Loughran (S.), « Configuration Description, Deployment and Lifecycle Management Working Group (CDDLM-WG) Final Report ». Rapport technique, HP, 2008.

[82] Goldsack (P.), Guijarro (J.), Lain (A.) *et al.*, « SmartFrog : Configuration and Automatic Ignition of Distributed Applications », dans *HP OpenView University Association conference (HP OVUA), Genève, Suisse*, juillet 2003.

[83] Commitee Draft 01 (O.), *Solution Deployment Descriptor Specification 1.0 Errata*. OASIS, novembre 2008.

[84] Object Management Group, « Deployment and Configuration of Component-based Distributed Applications Specification, version 4.0 ». Available Specification n° formal/2006-04-02, OMG, avril 2006.

[85] Marvie (R.), Merle (P.), Geib (J.-M.) et Vadet (M.), « OpenCCM : une plate-forme ouverte pour composants CORBA », dans *2ème Conférence Française sur les Systèmes d'Exploitation (CFSE), Paris, France*, p. 112, avril 2001.

[86] Baude (F.), Caromel (D.), Mestre (L.) *et al.*, « Interactive and Descriptor-based Deployment of Object-Oriented Grid Applications », dans *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, Edinburgh, Scotland*, p. 93–102. IEEE Computer Society, juillet 2002.

[87] Depalma (N.), Bouchenak (S.), Boyer (F.) *et al.*, « Jade : un environnement d'administration autonome », *Technique et Science Informatiques*, vol. 27, n° 9-10, 2008, p. 1225–1252.

[88] Broto (L.), Hagimont (D.), Stolf (P.) *et al.*, « Autonomic management policy specification in Tune », dans *Annual ACM Symposium on Applied Computing (SAC'08), Fortaleza, Ceara, Brazil*, p. 1658–1663. ACM, mars 2008.

[89] Flissi (A.), Dubus (J.), Dolet (N.) et Merle (P.), « Deploying on the Grid with DeployWare », dans *Proceedings of the 8th International Symposium on Cluster Computing and the Grid (CCGRID'08), Lyon, France*, p. 177–184. IEEE, mai 2008.

[90] Caron (E.), Chouhan (P. K.) et Dail (H.), « GoDIET : A Deployment Tool for Distributed Middleware on Grid'5000 », dans IEEE, éditeur, *EXPGRID Workshop, Paris, France. Experimental Grid Testbeds for the Assessment of Large-Scale Distributed Apllications and Tools. In conjunction with HPDC-15.*, p. 1–8, juin 2006.

[91] Chouhan (P. K.), *Automatic Deployment for Application Service Provider Environments*. Thèse de doctorat, École Normale Supérieure de Lyon, septembre 2006.

[92] Lacour (S.), *Contribution à l'automatisation du déploiement d'applications sur des grilles de calcul*. Thèse de doctorat, Université de Rennes I, décembre 2005.

[93] Bolze (R.), Cappello (F.), Caron (E.) *et al.*, « Grid'5000 : A Large Scale and Highly Reconfigurable Experimental Grid Testbed », *International Journal of High Performance Computing Applications*, vol. 20, n° 4, novembre 2006, p. 481–494.

[94] Capit (N.), Costa (G. D.), Georgiou (Y.) *et al.*, « A Batch Scheduler with High Level Components », dans *Proceedings of Cluster Computing Grid 2005 (CCGRID'05)*, p. 776–783, 2005.

[95] Cudennec (L.), *CoRDAGe : un service générique de co-déploiement et redéploiement d'applications sur grilles*. Thèse de doctorat, Université de Rennes I, janvier 2009.

[96] Simon (H. A.), *Les sciences de l'artificiel*, chap. L'intelligibilité des mondes naturels et artificiels, p. 33–34. Gallimard, janvier 2004. traduit par Jean-Louis Le Moigne.

[97] Kuchling (A.), van Rossum (G.), Jr (F. L. D.) et Ward (G.), *Python 2.5 Documentation*. Python Software Foundation, septembre 2006.

[98] Thenault (S.), *Pylint User Manuel*. Logilab, septembre 2009.

[99] Boyd (L. L.), « Patterns of Event Objects in Large Scale Business Systems », dans *Proccedings of EuroPLoP'96*, juin 1996.

[100] Doctrine-project.org, *Join Table Inheritance*, octobre 2008.

[101] Gamma (E.), Helm (R.), Johnson (R.) et Vlissides (J.), *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[102] Lee (J.). « Python Actor Runtime Library ». http ://osl.cs.uiuc.edu/parley/.

[103] Ribes (A.) et Caremoli (C.), « SALOME platform component model for numerical simulation », dans *COMPSAC (2)*, p. 553–564. IEEE Computer Society, 2007.

[104] Pérez (C.), Priol (T.) et Ribes (A.), « A Parallel CORBA Component Model for Numerical Code Coupling », *International Journal of High Performance Computing Applications (IJHPCA)*, vol. 17, n° 4, 2003, p. 417–429. Special issue Best Applications Papers from the 3rd International Workshop on Grid Computing.

[105] Wienberg (A.), Matthes (F.) et Boger (M.), « Modeling Dynamic Software Components in UML », dans *UML99 - The Unified Modeling Language. Proceedings, LNCS 1723*, p. 204–219. Springer-Verlag, 1999.